НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

(повне найменування інституту, факультету)

Автоматизованих систем обробки інформації і управління

(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)
“ ” 2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки ***6.050103 «Програмна інженерія»***
на тему *Платформа для побудови застосунків з обміну геоданими*

Виконав: студент IV курсу, групи *ІП-51 Кліщ Олег Володимирович* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *ас. Очеретяний О.К.* _____
посада,науковий ступінь,вчене звання,прізвище,ініціали (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.* _____
посада,науковий ступінь,вчене звання,прізвище,ініціали (підпис)

Рецензент:

_____
посада,науковий ступінь,вчене звання,прізвище,ініціали (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки (програма професійного спрямування) – *6.050103 «Програмна інженерія» (Програмне забезпечення систем)*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *О.А. Павлов*
(підпис) (ініціали, прізвище)
“___”_____2019 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТУ

Кліщу Олегу Володимировичу
(прізвище, ім’я, по батькові)

1. Тема проекту «*Платформа для побудови застосунків з обміну геоданими*»
керівник проекту *Очеретяний Олександр Констянтинович, асистент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “23” квітня 2019 р. №1181-с

2. Термін подання студентом проекту «03» *червня 2019 року*

3. Вихідні дані до проекту
Технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, архітектура програмного забезпечення, аналіз безпеки даних
3) Аналіз якості та тестування програмного забезпечення: опис процесів та

контрольного прикладу

4) Впровадження та супровід програмного забезпечення: розгортання та робота з програмним забезпеченням

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема бази даних

3) Схема структурна бізнес процесів

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «15» *квітня 2019 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	*Вивчення рекомендованої літератури*	*20.04.2019*	
2.	*Аналіз існуючих методів розв'язання задачі*	*20.04.2019*	
3.	*Постановка та формалізація задачі*	*23.04.2019*	
4.	*Аналіз вимог до програмного забезпечення*	*25.04.2019*	
5.	*Алгоритмізація задачі*	*29.04.2019*	
6.	*Моделювання програмного забезпечення*	*30.04.2019*	
7.	*Обґрунтування використовуваних технічних засобів*	*05.05.2019*	
8.	*Розробка архітектури програмного забезпечення*	*11.05.2019*	
9.	*Розробка програмного забезпечення*	*16.05.2019*	
10.	*Налагодження програми*	*20.05.2019*	
11.	*Виконання графічних документів*	*24.05.2019*	
12.	*Оформлення пояснювальної записки*	*27.05.2019*	
13.	*Подання ДП на попередній захист*	*28.05.2019*	
14.	*Подання ДП рецензенту*	*03.05.2019*	
15.	*Подання ДП на основний захист*	*08.06.2019*	

Студент _____ Кліщ О.В.
(підпис)

Керівник проекту _____ Очеретяний О.К.
(підпис)

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	КПІ.ІП-5107. 045430.02.81	Пояснювальна записка	61	
3	А4	КПІ.ІП-5107. 045430.03.91	Технічне завдання	11	
4	А4	КПІ.ІП-5107. 045430.04.51	Програма та методика тестування	2	
5	А4	КПІ.ІП-5107. 045430.05.34	Керівництво адміністратора	3	
6	А4	КПІ.ІП-5107. 045430.06.34	Керівництво користувача	9	
7	А4	КПІ.ІП-5107. 045430.07.13	Опис програми	16	
8	А3	КПІ.ІП-5107. 045430.08.99	Схема структурна варіантів використання	1	
9	А3	КПІ.ІП-5107. 045430.08.99	Схема структурна бази даних	1	
10	А3	КПІ.ІП-5107. 045430.08.99	Схема структурна бізнес процесів	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-5107.045430.00.90		
Розробн.		Кліщ О.В.			Відомість дипломного проекту	Літ.	Лист 1 / Листів 1
Керівн.		Очеретяний О.К.					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-51	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

АНОТАЦІЯ

Пояснювальна записка складається із 4 розділів та містить 14 рисунків, 39 таблиць, 6 додатків та 10 джерел – загалом 108 сторінок.

Метою проекту є розробка платформи, яка б дозволяла IT-підприємцям створювати сервіси, що працюють із геоданими, як-от застосунки по відстеженню кур'єра із замовленням чи пошук громадського транспорту. Розробка сервісу в платформі зводиться до описання бізнес-логіки застосунку у декларативній формі без необхідності інвестувати ресурси в розробку застосунку з нуля. Користувачі сервісу взаємодіють із ним за допомогою мобільного застосунку, який надається платформою і є універсальним для всіх сервісів.

У розділі «Аналіз вимог до програмного забезпечення» описано предметну область, проаналізовано існуючі технічні рішення та успішні програмні продукти. Також розроблено функціональні та нефункціональні вимоги.

У розділі «Моделювання та конструювання програмного забезпечення» було описано бізнес процеси та розроблено архітектуру платформи. В даному розділі описано такі складові серверного програмного забезпечення, як-от авторизація, зберігання даних та спосіб опису бізнес-логіки сервісу. Також було описано основні частини мобільного застосунку.

У розділі «Аналіз якості та тестування програмного забезпечення» описано процеси тестування та основні сценарії тестування платформи.

У розділі «Впровадження та супровід програмного забезпечення» описано процес розгортання сервера та мобільного застосунку.

Ключові слова: ГЕОДАНІ, ПЛАТФОРМА, ПОБУДОВА ЗАСТОСУНКІВ, БІЗНЕС-ЛОГІКА, ХМАРНІ ОБЧИСЛЕННЯ.

ABSTRACT

The explanatory note consists of 4 sections and contains 14 figures, 39 tables, 6 appendixes and 10 sources - 108 pages in total.

The goal of the project is to develop a platform that would allow IT-entrepreneurs to create services that work with geodata, such as courier tracking software or applications for public transport search. One can develop service in the platform by describing its business logic in declarative form without the need to invest resources in the development of the application from scratch. Service users interact with it using a mobile application that is provided by the platform and is universal for all services.

In the section "Analysis of Software Requirements" described the subject, analyzed the existing technical solutions and successful products. Functional and non-functional requirements were also developed here.

In the section "Modeling and Designing Software" described business processes and developed the platform architecture. Also in the section outlined different parts of the server side, such as authorization, data storage, and a way to describe the business logic of the service. The main parts of the mobile application were also described.

In the section "Analysis of the quality and testing of software" described the testing process and main scenarios.

In the section "Introduction and Software Support" described how to deploy the server and mobile application.

Keywords: GEODATA, PLATFORM, APPLICATION BUILDER, BUSINESS LOGIC, CLOUD COMPUTING.

Пояснювальна записка
до дипломного проекту

на тему: «Платформа для побудови застосунків з обміну геоданими»

Київ – 2019 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ПОЗНАЧЕНЬ

DSL (Domain specific language) – комп'ютерна мова, що спеціалізується на певній області застосування.

NoSQL (not only SQL) - ряд підходів спрямованих на реалізацію СКБД, що відмінні від традиційних реляційних, задля вирішення проблем масштабованості та доступності за рахунок атомарності та узгодженості.

REST (Representational State Transfer) – архітектурний підхід до розробки інтерфейсу інформаційних систем.

JWT (JSON Web Token) – відкритий стандарт для створення токенів доступу до інформаційних систем, що оснований на форматі JSON.

Activity – компонент застосунків на операційній системі Android, що має графічний інтерфейс та дозволяє користувачам виконувати дії, як-от подивитися на карту чи відправити лист.

ВСТУП

Обмін даних є центральною частиною багатьох програмних продуктів. Наприклад, месенджери дозволяють обмінюватися повідомленнями, зрозумілими для людей, із друзями і рідними. Торрент-трекери надають можливість напряму обмінюватися файлами користувачам мережі.

Програмні продукти в яких обмін геоданими відіграє ключову роль появляються все частіше. Прикладом є популярний сервіс Uber, що дозволяє пасажирам знайти таксі і відстежувати його місцезнаходження під час очікування. Іншим хорошим прикладом є функція обміну координати в Facebook Messenger, що допомагає людям знайти один одного в незнайомому для них місці. Також обмін геоданими допомагає батькам слідкувати за їх дітьми під час подорожей за місто із класом.

Окрім програмних продуктів для обміну геоданими, є ще один дуже цікавий тренд — хмарні технології. Такий сервіс, як-от Amazon Web Services (AWS) дозволяє IT-підприємцях будувати складні програмні продукти без додаткових інвестицій в серверну інфраструктуру, її адміністрування та покупку ліцензій. Це значно знижує поріг входу в IT-бізнес. Ще будучи стартапом, клієнтом AWS був Instagram, який сьогодні має мільйони користувачів.

Аналізуючи два вищезгадані тренди, з’являється ідея в створенні універсальної платформи для побудови, розгортання та адміністрування сервісів, де ключову роль відіграє обмін геоданими. IT-підприємець, описавши бізнес-логіку в декларативний спосіб, отримує готовий сервіс доступний із інтернету для його користувачів. Така платформа знизить поріг входу в цей сегмент IT-ринку, що збільшить кількість, і як результат якість, програмних продуктів на ринку.

Метою даної роботи є розробка універсальної платформи для побудови сервісів, де ключову роль відіграє обмін геоданих. Це включає розробку серверу із REST API для створення сервісу, DSL (Domain Specific Language) для опису його бізнес-логіки та універсального мобільного застосунку для обміну

геоданих, що будуть автоматично підлаштовуватися під бізнес-логіку конкретного сервісу.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Кількість продуктів на ринку та рівень їх інноваційності обернено пропорційний мінімальній кількості ресурсів, які треба на початковому етапі для отримання перших значних результатів. Чим необхідна кількість ресурсів менша, тим більше буде бажаючих, і тим швидше розвиватиметься ринок.

Основна причина високого порогу входу — необхідність реалізовувати велику кількість речей, що не дають прямого результату. Це може бути вибір стеку технологій, побудова архітектури, написання основного каркасу програмного забезпечення. Один раз спроектований код, із можливістю конфігурації за допомогою певної декларативної мови, може покрити більшість типових задач, які стоять перед бізнесом на початковій стадії.

Платформою для побудови застосунків по обміну геоданими будемо називати набір серверного та мобільного програмного забезпечення, які можна налаштувати під нові бізнес задачі, і таким чином отримати новий сервіс. Бізнес-логіка, яку можна описати в такій платформі, обмежена роботою із геоданими. В межах однієї платформи можуть паралельно існувати декілька сервісів. Всі вони ізольовані один від одного.

1.2 Змістовний опис і аналіз предметної області

Геодані — це набір точок, де кожна задається широтою і довготою. Додатково кожна точка може мати мета-дані. Метадані для точки — це список пар ключ-значення, що використовуються для побудови бізнес-логіки застосунку. Наприклад, в сервісі доставки піци, точка на карті представляє кур’єра. Мета-даними для кур’єра може бути номер поточного замовлення за допомогою якого клієнт матиме можливість його знайти.

Бізнес-логіка застосунку задається в JSON форматі і складається із набору ролей. Роль задається набором метаданих, які може поширювати користувач

разом із геоданими та списку можливих запитів. Запит — це предикат, що оперує значенням координат та метаданими точки. Всі точки для яких предикат повернув істину повертаються як результат запиту. Підставляти константні значення в код запиту можна з мобільного застосунку за допомогою механізму змінних. Наведемо приклад запиту, що повертає дані про кур'єра – «orderNo EQ %orderNo». Вираз «orderNo» - це назва тегу (пари з метаданих), а «%orderNo» — змінна, що буде встановлена користувачем, який виконуватиме запит.

Універсальний мобільний застосунок отримує опис обраного сервісу і показує доступні ролі. За допомогою графічного інтерфейсу можна створити новий обліковий запис із певною роллю чи увійти під існуючим. Можливі розділи програми будуються автоматично з опису ролі користувача, що авторизувався. Якщо роль передбачає поширення даних, то по замовчуванню можна поширювати координату. Набір мета-даних, які можна додатково поширювати задається в описі ролі. Такий самий підхід використовується для запитів. Якщо в опису ролі є хоча хоча б один запит, то він буде відображено в графічному інтерфейсі. Щоб скористатися запитом потрібно вказати значення всіх змінних, які у ньому використовуються. Після цього результат запиту відображатиметься в реальному часі в розділі «Карта».

1.3 Аналіз успішних ІТ-проектів

1.3.1 Аналіз відомих технічних рішень

Ключовим принципом побудови подібних систем є модель SaaS (Software as a service). Це такий підхід при якому клієнт не володіє програмним забезпеченням, а платить за сам факт його використання. Часто такі сервіси надають, як інструменти для створення прикладних рішень, так і інфраструктуру для їх розгортання. Прикладом таких сервісів є конструктори веб-сайтів. Вони дозволяють створити базовий веб-сайт без знань програмування, і одночасно надають домен і хостинг. Перевагами SaaS є суттєве скорочення витрат на розгортання, адміністрування сервісу та висока швидкість впровадження. Ще

однією важливою складовою платформи, що розробляється в даній роботі, є опис бізнес-логіки сервісу. Для цього потрібно розробити власну мову враховуючи знання домену (в нашому випадку це геодані). Такі мови називаються предметно-орієнтованими (Domain-Specific). Прикладами є SQL (Structured Query Language), що використовується для роботи з даними в СУБД, HTML (Hypertext Markup Language) — мова опису вмісту веб-сторінок. Для спрощення процесу опису рішення часто використовують декларативні мови, бо людині простіше вказати, що вона хоче, ніж як це отримати.

Доступ до функцій серверу часто реалізується через REST API (Representational State Transfer Application Interface). Це означає відсутність стану, специфічного для конкретного користувача, на сервері, що дозволяє масштабуватися горизонтально. Також плюсом REST API є його простота і поширеність. Це дає можливість стороннім розробникам створювати власні клієнтські застосунки, що працюватимуть із сервером по даному прикладному протоколу.

Отже, набір існуючих технічних рішень, що було наведено вище, є корисним для розробки універсальної платформи по обміну геоданих.

1.3.2 Аналіз відомих програмних продуктів

Після аналізу ринку та пошуку подібних програмний продуктів було знайдено наступні рішення для обміну геоданими:

- Spyzie;
- Facebook Messenger;
- Attendify.

Spyzie – це сервіс, що дозволяє батькам слідкувати за їх дітьми. Spyzie відслідковує різноманітну інформацію з Android пристроїв. Наприклад, як-от географічні координати, всі отримані та надіслані повідомлення і історію дзвінків. Тобто, обмін геоданими між телефоном дитини і веб-інтерфейсом батьків є лише однією із можливостей сервісу. Spyzie працює по моделі SaaS і

продає підписки для користування сервісом на певний період. Кількість параметрів, які можна відслідковувати за допомогою даного рішення є його великою перевагою. Основним мінусом є те, що в Spyzie не можна задати логіку реагування на зібрані дані. На рисунку 1.1 зображено головне вікно Spyzie.

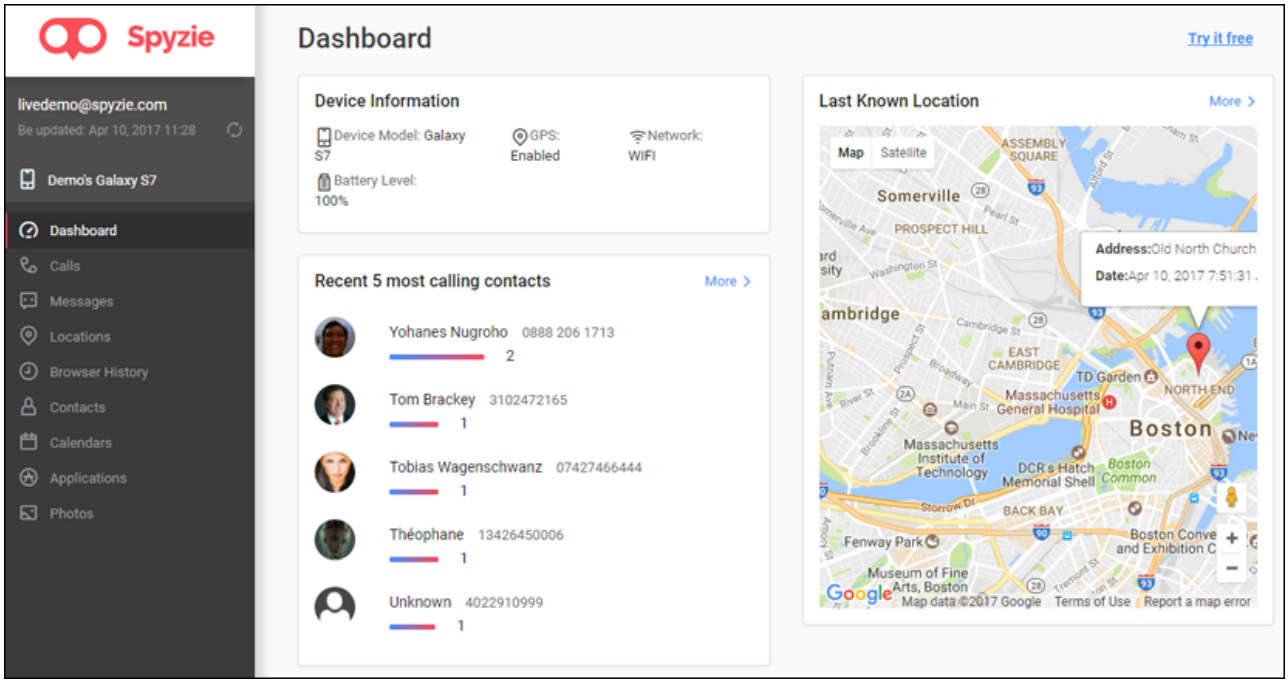

Рисунок 1.1 Інтерфейс Spyzie

Facebook Messenger має функцію обміну координат в реальному часі. Це корисний функціонал, який використовується для координації друзів між собою. Це не є самодостатнім сервісом, а лише корисним доповненням до мессенджеру. Користуватися ним просто. Перевагами є інтегрованість в екосистему Facebook, безплатне користування. Проте, ця функціональність не є окремим самодостатнім продуктом і підходить тільки для декількох варіантів використання. На рисунку 1.2 зображено процес обміну координатами в реальному часу між двома друзями, що намагаються знайти один одного.

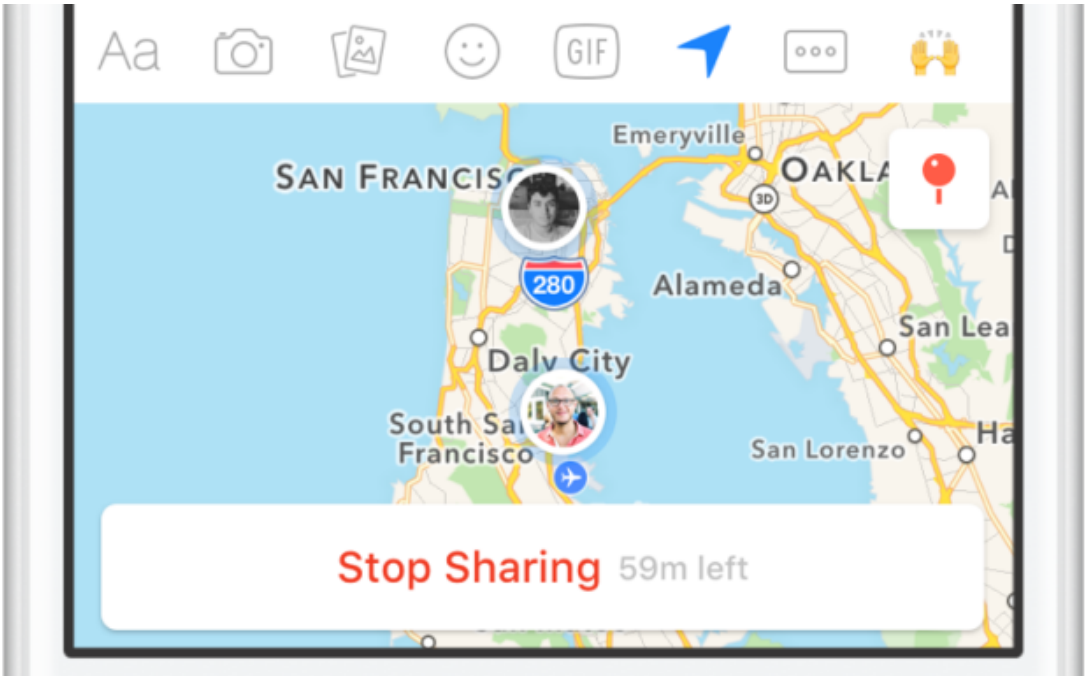

Рисунок 1.2 Поширення координати в Facebook Messenger

Attendify — це сервіс, що дозволяє планувати конференції, виставки та інші події. Однією із можливостей є створення інтерактивних карт заходу. Правда єдині карти, які можна створювати - статичні. Як і Spyzie, Attendify побудований по моделі SaaS, але оплата прив'язана не до періоду часу, а до кількості відвідувачів події. Перевагою сервісу є велика кількість інструментів для організації подій. Але в контексті рішення, яке ми шукаємо, даний продукт немає достатнього функціоналу по роботу із картами.

Spyzie та Facebook Messenger дозволяють обмінюватися геоданими між декількома групами людей, але вони не дозволяють додавати бізнес-логіки по роботі із цими даними. Attendify має відношення до роботи геоданими, але це не є його основною частиною і проблему, яку він вирішує є планування заходів.

Отже, доступні рішення не є достатньо гнучкі, щоб дозволити створення застосунків, що не тільки обмінюються геоданими, а й виконують додаткову бізнес-логіку сервісу.

Після аналізу існуючих програмних рішень, можемо зробити висновок, що розробка платформи, яка б дозволяла створювати сервіси із власною бізнес-логікою по роботі з геоданими є актуальною.

1.4 Аналіз вимог до програмного забезпечення

В системі є дві ролі: адміністратор та користувач. Адміністратор створює сервіс, задає бізнес-логіку та займається внесенням необхідних змін. Користувач використовує сервіс, створений адміністратором, для поширення та/або отримання геоданих.

Зручним і часто використовуваним способом задання вимог є діаграми використання. На рисунку 1.3 наведення діаграму використання для розроблюваної системи.

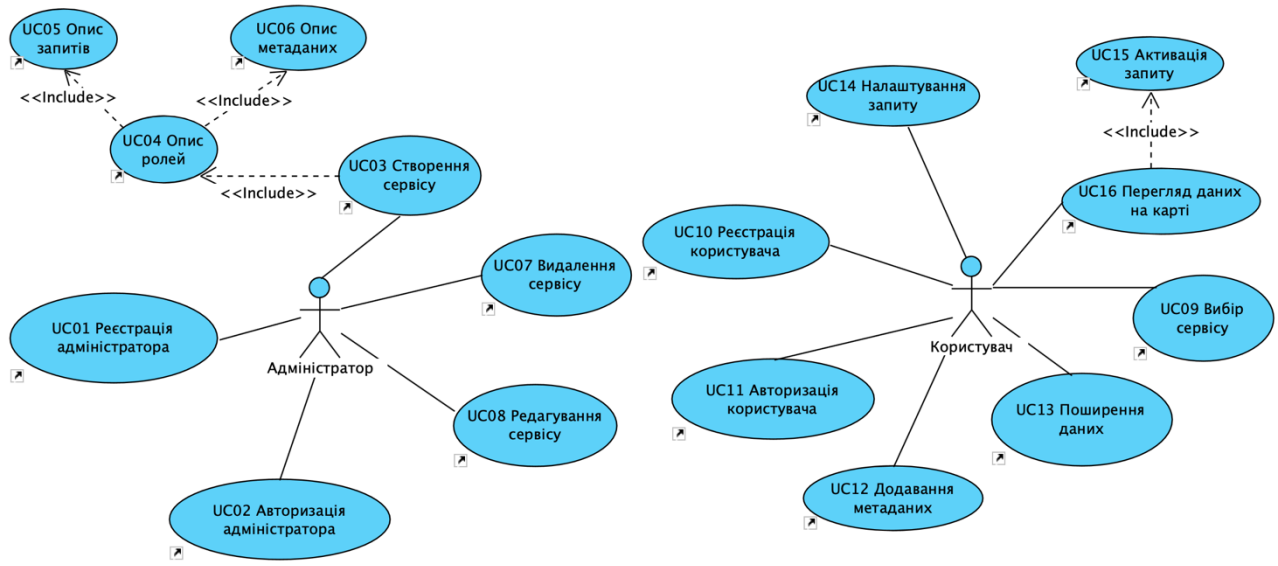

Рисунок 1.3 – Схема структура варіантів використання

Для кожного варіанту використання наведено таблицю із ідентифікатором використання, найменуванням, актором, описом, передумовами, вихідними умовами та виключними ситуаціями. Розглянемо кожен випадок детально.

Таблиця 1.1 – Варіант використання «Реєстрація адміністратора»

Ідентифікатор	UC01
Найменування	Реєстрація адміністратора
Актор	Адміністратор
Опис	Адміністратор реєструється на платформі
Попередні умови	
Вихідні умови	Створено новий обліковий запис адміністратора

Таблиця 1.2 – Варіант використання «Авторизація адміністратора»

Ідентифікатор	UC02
Найменування	Авторизація адміністратора
Актор	Адміністратор
Опис	Адміністратор авторизується в системі
Попередні умови	Адміністратор зареєстрований
Вихідні умови	Адміністратор отримав авторизаційний токен

Таблиця 1.3 – Варіант використання «Створення сервісу»

Ідентифікатор	UC03
Найменування	Створення сервісу
Актор	Адміністратор
Опис	Адміністратор створює сервіс, описавши його бізнес-логіку
Попередні умови	Адміністратор авторизований
Вихідні умови	Сервіс створено

Таблиця 1.4 – Варіант використання «Опис ролей»

Ідентифікатор	UC04
Найменування	Опис ролей
Актор	Адміністратор
Опис	Адміністратор описує бізнес-логіку сервісу додаючи туди ролі користувачів
Попередні умови	Адміністратор авторизований
Вихідні умови	Ролі додано до сервісу

Таблиця 1.5 – Варіант використання «Опис запитів»

Ідентифікатор	UC05
Найменування	Опис запитів
Актор	Адміністратор
Опис	Адміністратор додає запит до ролі, задавши предикат
Попередні умови	Адміністратор авторизований
Вихідні умови	Запит додано до ролі

Таблиця 1.6 – Варіант використання «Опис метаданих»

Ідентифікатор	UC06
Найменування	Опис метаданих
Актор	Адміністратор

Продовження таблиці 1.6

Опис	Адміністратор задає список метаданих доступних для поширення разом із координатою для певної ролі користувачів сервісу
Попередні умови	Адміністратор авторизований
Вихідні умови	Дозволені метадані додано до ролі

Таблиця 1.7 – Варіант використання «Видалення сервісу»

Ідентифікатор	UC07
Найменування	Видалення сервісу
Актор	Адміністратор
Опис	Адміністратор видаляє сервіс
Попередні умови	Адміністратор авторизований та сервіс існує
Вихідні умови	Сервіс видалено

Таблиця 1.8 – Варіант використання «Редагування сервісу»

Ідентифікатор	UC08
Найменування	Редагування сервісу
Актор	Адміністратор
Опис	Адміністратор змінює бізнес-логіку існуючого сервіс
Попередні умови	Адміністратор авторизований та сервіс існує
Вихідні умови	Сервіс оновлено

Таблиця 1.9 – Варіант використання «Вибір сервісу»

Ідентифікатор	UC09
Найменування	Вибір сервісу
Актор	Користувач
Опис	Користувач обрає сервіс
Попередні умови	Сервіс існує
Вихідні умови	Сервіс вибрано

Таблиця 1.10 – Варіант використання «Реєстрація користувача»

Ідентифікатор	UC10
Найменування	Реєстрація користувача
Актор	Користувач
Опис	Користувач створює новий обліковий запис користувача із заданою роллю в межах сервісу
Попередні умови	Сервіс вибрано
Вихідні умови	Створено новий обліковий запис користувача

Таблиця 1.11 – Варіант використання «Авторизація користувача»

Ідентифікатор	UC11
Найменування	Авторизація користувача
Актор	Користувач
Опис	Користувач авторизується в системі
Попередні умови	Користувач зареєстрований
Вихідні умови	Користувач отримав авторизаційний токен

Таблиця 1.12 – Варіант використання «Додавання метаданих»

Ідентифікатор	UC12
Найменування	Додавання метаданих
Актор	Користувач
Опис	Користувач надає додаткові дані, що поширюватимуться разом із координатою
Попередні умови	Користувач авторизований
Вихідні умови	Метадані додано до координати

Таблиця 1.13 – Варіант використання «Поширення даних»

Ідентифікатор	UC13
Найменування	Поширення даних
Актор	Користувач
Опис	Користувач поширює дані в межах сервісу
Попередні умови	Користувач авторизований
Вихідні умови	Користувач поширив дані

Таблиця 1.14 – Варіант використання «Налаштування запиту»

Ідентифікатор	UC14
Найменування	Налаштування запиту
Актор	Користувач
Опис	Користувач вказує значення змінних, які необхідні для роботи запиту
Попередні умови	Користувач авторизований, запит доступний для користувача
Вихідні умови	Користувач налаштував запит для роботи

Таблиця 1.15 – Варіант використання «Активація запиту»

Ідентифікатор	UC15
Найменування	Активація запиту

Продовження таблиці 1.15

Актор	Користувач
Опис	Користувач активує запит, що означає відображення його результатів на карті
Попередні умови	Користувач авторизований, запит доступний для користувача, запит налаштований
Вихідні умови	Результати запиту додані на карту

Таблиця 1.16 – Варіант використання «Перегляд даних на карті»

Ідентифікатор	UC16
Найменування	Перегляд даних на карті
Актор	Користувач
Опис	Користувач переглядає дані на карті
Попередні умови	Користувач авторизований, запити доступні для користувача
Вихідні умови	Відображена карта із даними

1.4.1 Розроблення функціональних вимог

На основі варіантів використання сформульовано наступні функціональні вимоги. Також на рисунку 1.4 зображено залежності між функціональними вимогами.

Таблиця 1.17 – Опис функціональних вимог

Ідентифікатор	**Опис**	**Пріоритет**
REQ001	Система має надавати можливість реєструватися адміністраторам в платформі	Високий
REQ002	Система має надавати можливість отримати авторизаційний токен адміністратору	Високий
REQ003	Система має надавати можливість адміністратору можливість створити сервіс	Високий
REQ004	Система має надавати адміністратору можливість змінювати бізнес-логіку сервісу	Середній

Продовження таблиці 1.17

REQ005	Система має надавати адміністратору можливість видаляти сервіс	Низький
REQ006	Система має дозволяти незареєстрованому користувачу отримати список ролей сервісу	Високий
REQ007	Система має дозволяти користувачу реєструватися під певною роллю в сервісі	Високий
REQ008	Система має дозволяти зареєстрованому користувачу отримати авторизаційний токен	Високий
REQ009	Система має дозволяти користувачу отримати доступні йому запити та метадані	Високий
REQ010	Система має дозволяти користувачу виконувати запити	Високий
REQ011	Система має дозволяти користувачу поширювати дані	Високий
REQ012	Система має дозволяти користувачу візуалізувати результати запитів	Середній

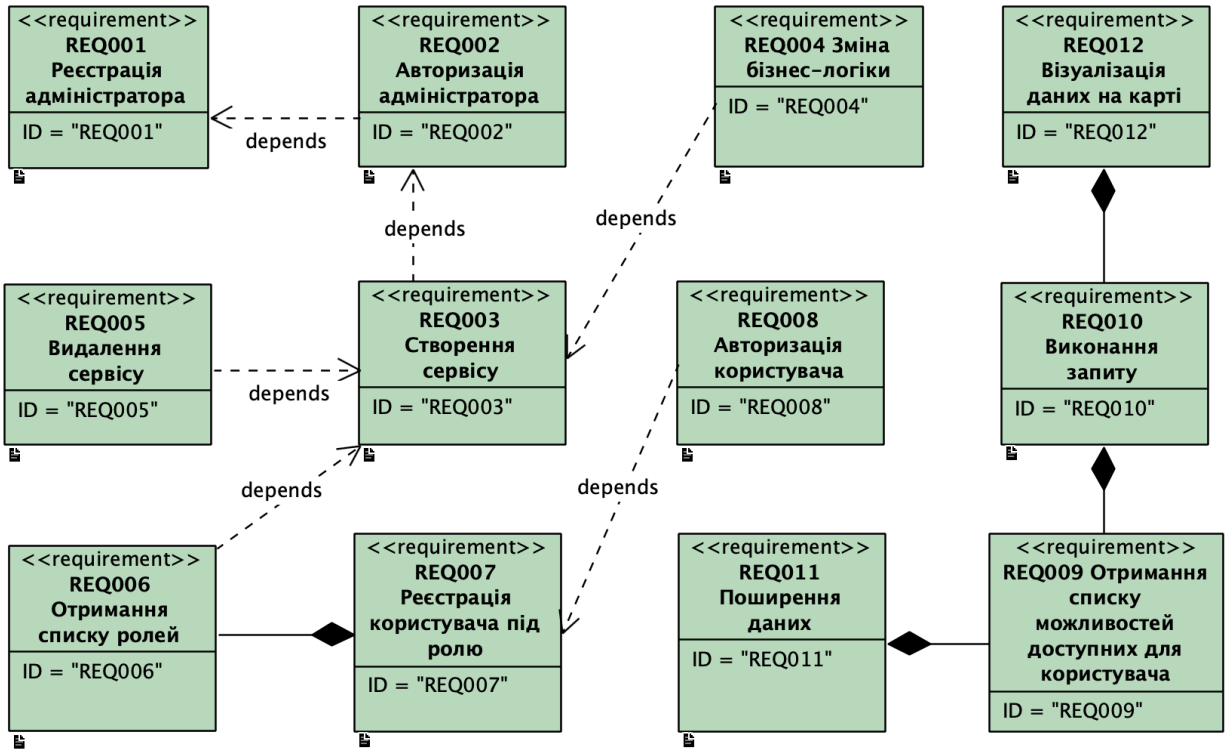

Рисунок 1.4 – Схема структурна функціональних вимог

На рисунку 1.5 зображено залежність між функціональними вимогами та сценаріями використання.

	UC01 Реєстрація адміністрат...	UC02 Авторизація адміністр...	UC03 Створення сервісу	UC04 Опис ролей	UC05 Опис запитів	UC06 Опис метаданих	UC07 Видалення сервісу	UC08 Редагування сервісу	UC09 Вибір сервісу	UC10 Реєстрація користувача	UC11 Авторизація користув...	UC12 Додавання метаданих	UC13 Поширення даних	UC14 Налаштування запиту	UC15 Активація запиту	UC16 Перегляд даних на ка...
REQ001 Реєстрація адміністратора	✓	✓														
REQ002 Авторизація адміністратора		✓	✓	✓	✓	✓	✓	✓				✓				✓
REQ003 Створення сервісу			✓	✓	✓	✓	✓	✓	✓	✓						
REQ004 Зміна бізнес-логіки								✓								
REQ005 Видалення сервісу							✓									
REQ006 Отримання списку ролей										✓						
REQ007 Реєстрація користувача під ролю										✓	✓					
REQ008 Авторизація користувача											✓		✓	✓	✓	
REQ009 Отримання списку можливостей до...												✓		✓	✓	✓
REQ010 Виконання запиту																✓
REQ011 Поширення даних												✓	✓			
REQ012 Візуалізація даних на карті																✓

Рисунок 1.5 – Матриця трасування

1.4.2 Розроблення нефункціональних вимог

Платформа для побудови застосувань по обміну геоданими складається із сервера та універсального мобільного застосунку.

Сервер надає REST API для керування сервісами. В серверну частину вбудований веб-сервер із документацію та утилітою для взаємодії із API. Обмін даними відбувається у форматі JSON. Авторизацію реалізована з використанням JSON Web Tokens. Для запуску коду потрібно JVM версії 8. Всі паролі мають зберігатися у хешованому вигляді. Дані сервісів мають бути ізольовані одне від одного.

Універсальний мобільний застосунок реалізований для платформи Android. Інтерфейс автоматично будується із конфігурації конкретного сервісу.

1.4.3 Постановка комплексу завдань модулю

Метою роботи є розробка платформи для створення застосунків по обміну геоданими. Платформа дозволить IT-підприємцям тестувати їх бізнес ідеї, описавши бізнес-логіку сервісу у декларативному стилі. На основі варіантів використання, функціональних та нефункціональних вимог можна виділити наступні основні вимоги:

- створення сервісу з описом його бізнес-логіки;
- управління обліковими записами користувачів;
- обмін і збереження даних в межах сервісу;
- пошук по даних доступних для конкретної ролі в межах сервісу;
- можливість створення конфігурацій на сервері.

1.5 Висновки по розділу

У даному розділі було описано сценарії використання системи, функціональні та нефункціональні вимоги. Було проаналізовано технологічні рішення, що успішно застосовувалися для вирішення конкретних підзадач, що містяться в даній роботі. Також розглянуто успішні програмні рішення із

подібним функціоналом. Проаналізувавши їх сильні та слабкі сторони було показано актуальність розробки платформи для побудови застосунків з обміну геоданими.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

На рисунку 2.1 наведена BPMN діаграма, що описує процес взаємодії користувачів із сервісом платформи. Перший користувач поширює дані, а другий виконує запит. Кожен користувач запитує у платформи набір можливостей. Для першого користувача це список метаданих, які потрібно встановити для поширення разом із координатою. Далі перший користувач вводить дані і починає надсилати їх платформі. Платформа для кожного користувача, що може поширювати дані, зберігає останні значення, що було отримано. Другий користувач запитує список запитів. Обравши один з них, вводить змінні, необхідні для виконання запиту. Наприклад, номер замовлення у запиті, що повертає дані про кур'єра. Отримавши дані, сервер виконує предикат над координатами і метаданими та повертає всі такі точки, що задовільними йому. Кожному користувачу, що може поширювати дані відповідає одна точка. Результат запиту другий користувач відображає на карті. Дані показують останнє значення на сервері та оновлюється у режимі реального часу.

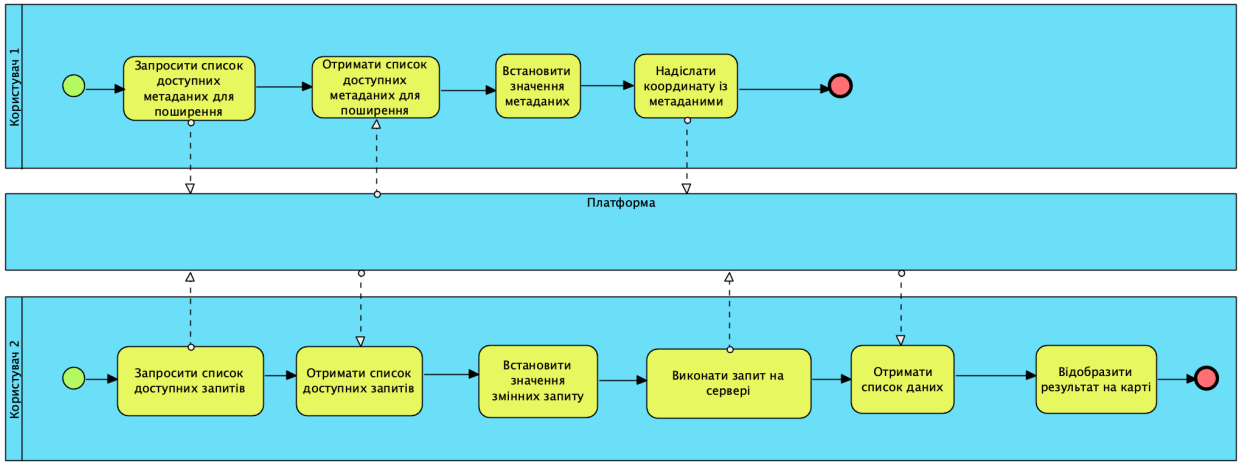

Рисунок 2.1 – Схема бізнес процесів

Користувачі взаємодіють із сервісом через мобільний застосунок, а адміністратор керує ним через REST за допомогою Swagger UI.

2.2 Архітектура програмного забезпечення

2.2.1 Опис модулів серверного коду

Серверний код складається із наступних модулів:

- com.acarus.geo.config – модуль, що містить конфігураційні класи;
- com.acarus.geo.constants – модуль, що містить набір сталих, що часто використовуються в різних частинах коду;
- com.acarus.geo.controller – модуль, що містить контролери;
- com.acarus.geo.dto – модуль, що містить вхідні та вихідні структури даних;
- com.acarus.geo.exceptions – модуль, що містить структури для представлення помилок виконання програми;
- com.acarus.geo.filter – модуль, що містить класи для інтерпретації власної предметно-орієнтованої мови;
- com.acarus.geo.init – модуль, що містить класи для ініціалізації системи на старті;
- com.acarus.geo.model – модуль, що містить структури даних, які відповідають таблицям в базі даних;
- com.acarus.geo.repository – модуль, що містить набір класів по роботі із базою даних;
- com.acarus.geo.reposity – модуль, що містить класи для управління доступом до інформаційних ресурсів;
- com.acarus.geo.service – модуль, що містить основну бізнес-логіку платформи;
- com.acarus.geo.utils – модуль, набір допоміжних класів.

Для виконання конкретних дій контролери звертаються до класів сервісного рівня. Саме у них у містить бізнес-логіка платформи. На рисунку 2.2 наведено діаграму класів сервісного рівня.

Рисунок 2.2 – Діаграма класів сервісного рівня

На рисунку 2.3 зображено діаграму класів обмеження доступу. Вони відповідають за видачу токенів доступу та їх перевірку. А також надають можливість перевірки наявності прав для роботи із конкретним методом.

Рисунок 2.3 – Діаграма класів по керуванню доступом

2.2.2 Інтерфейс серверної частини

Доступ до інформаційних ресурсів серверу надається за допомогою REST API. Перевагами даного підходу є те, що він широко використовується в індустрії і дозволяє стороннім розробникам реалізувати власний клієнтський застосунок. Дані між сервером та клієнтом передаються у форматі JSON. Програмний інтерфейс структурований по функціональному призначенню у наступних контролерах:

– AdminController - реєстрація адміністраторів, що створюватимуть сервіси та керуватимуть ними;

– AdminAuthController - автентифікація адміністраторів;

– ServiceController - створення і редагування сервісів, а також отримання інформації про них;

– ServiceUserController - реєстрація користувачів в межах конкретного сервісу та отримання інформації по них;

– ServiceAuthController - автентифікація користувачів в межах сервісу;

– ServiceDataController - поширення даних та виконання запитів в межах сервісу.

В REST аргументи можуть передаватися, як у тілі запиту так і в URI. Наприклад, для створення користувача передають JSON структуру в тілі запиту, а для отримання інформації про нього поміщають ідентифікатор в URI. Останнє робиться для того, щоб відобразити в адресі запиту над яким саме ресурсом виконуватиметься метод. В таблицях 2.1 — 2.6 наведено опис методів кожного з контролерів, що складається із адреси запиту (URI), HTTP метода, призначення, вхідної та вихідної структури даних. Детальний опис структур даних наведений в таблиці 2.7. URI-аргументи вказані в адресі запиту у фігурних дужках, а їх опис наведений в таблиці 2.8.

Таблиця 2.1 – Методи AdminController

URI	**Метод**	**Вхідний тип**	**Вихідний тип**	**Призначення методу**
/admin	POST	AdminPostDto	AdminGetDto	Створити адміністратора
/admin	GET		AdminGetDto	Отримати інформацію про адміністратора, що викликав даний метод
/admin	DELETE			Видалити обліковий запис адміністратора, що викликав даний метод

Таблиця 2.2 – Методи AuthController

URI	**Метод**	**Вхідний тип**	**Вихідний тип**	**Призначення методу**
/auth/login	POST	LoginRequestDto	LoginResponse Dto	Автентифікувати адміністратора та надати йому JWT токен

Таблиця 2.3 – Методи ServiceController

URI	**Метод**	**Вхідний тип**	**Вихідний тип**	**Призначення методу**
/service	POST	ServiceDefinition PostDto	ServiceDefiniti onGetDto	Створити сервіс
/service	PUT	ServiceDefinition PostDto	ServiceDefiniti onGetDto	Повністю оновити сервіс

Продовження таблиці 2.3

/service/{serviceId}	DELETE			Видалити сервіс за ідентифікатором
/service/{serviceId}	GET		ServiceDefinitionGetDto	Отримати повний опис сервісу, доступний для адміністратора
/service/{serviceId}/info	GET		ServiceInfoDto	Отримати частковий опис сервісу, доступний публічно

Таблиця 2.4 – Методи ServiceUserController

URI	**Метод**	**Вхідний тип**	**Вихідний тип**	**Призначення методу**
/service/{serviceId}/user	POST	ServiceUserPostDto	ServiceUserGetDto	Створити користувача сервісу
/service/{serviceId}/user	DELETE			Видалити обліковий запис користувача сервісу

Продовження таблиці 2.4

/service/{serviceId}/user/info	GET		ServiceUserInfoDto	Отримати список можливостей користувача в межах заданого сервісу

Таблиця 2.5 – Методи ServiceAuthController

URI	**Метод**	**Вхідний тип**	**Вихідний тип**	**Призначення методу**
/service/{serviceId}/auth/login	POST	LogInRequestDto	LogInResponseDto	Автентифікувати користувача сервісу та видати JWT токен

Таблиця 2.6 – Методи ServiceDataController

URI	**Метод**	**Вхідний тип**	**Вихідний тип**	**Призначення методу**
/service/{serviceId}/data	POST	DataPostDto		Поділитися даними в межах сервісу
/service/{serviceId}/data/request	POST	DataRequestDto	Масив DataPostDto	Виконати запит по даних в межах сервісу

Таблиця 2.7 – Структури даних

Назва	**Структура**	**Опис**

AdminPostDto	{
	"password": string,
	"username": string
	}
		Інформація для реєстрації облікового запису адміністратора, що складається із імені та паролю

AdminGetDto	{
	"id": integer,
	"username": string
	}
		Інформація про зареєстрованого користувача, що складається із ідентифікатора та імені

LogInRequestDto	{
	"password": string,
	"username": string
	}
		Інформація потрібна для автентифікації, що складається із імені користувача та паролю

LogInResponseDto	{
	"exp": long,
	"token": string
	}
		Інормація про успішну аутентифікацію, що складається із JWT токена та терміну його дії

Продовження таблиці 2.7

ServiceDefinitionPostDto	{
	"name": string,
	"roles": [
	{
	"name": string,
	"properties": {
	"canShareData": boolean
	},
	"queries": [
	{
	"desc": string,
	"expr": string,
	"variables": [
	{
	"desc": string,
	"name": string
	}
	]
	}
	],
	"tags": [
	{
	"defaultValue":
		Опис бізнес-логіки сервісу для його створення в платформі. Опис складається із назви, списку опису ролей та URI адреси сервісу в межах платформи. Опис ролі складається із назви, булевого значення, що вказує на дозвіл поширення даних, списку запитів та списку тегів. У свою чергу запит складається із опису, тіла запиту та списку змінних, що

Продовження таблиці 2.7

	string, "desc": string, "name": string, "required": boolean }] }], "uri": string }	передаються в запит. Опис тегу складається із назви, опису, значення по замовчуванню та булевого значення, що вказує чи тег є обов'язковим для поширення.

ServiceDefinitionGetDto	{
	"id": long,
	"name": string,
	"roles": [
	{
	"id": long,
	"name": string,
	"properties": {
	canShareData": boolean
	},
	"queries": [
		Повний опис уже створеного сервісу. Відрізняється від ServiceDefinitionPostDto лише тим, що у кожної сутності наявний ідентифікатор, що був згенерований сервером при сервісу.

Продовження таблиці 2.7

	"
	{
	"desc": string,
	"expr": string,
	"id": long,
	"variables": [
	{
	"desc": string,
	"name": string
	}
	]
	}
	],
	"tags": [
	{
	"defaultValue": string,
	"desc": string,
	"id": long,
	"name": string,
	"required": boolean
	}
	]
	}	

Продовження таблиці 2.7

	], "uri": string }	

ServiceInfoDto	{
	"id": long,
	"name": string,
	"roles": [
	{
	"id": long,
	"name": string
	}
	],
	"uri": string
	}
		Короткий опису сервісу для публічного доступу. Опис складається із ідентифікатора сервісу, імені, адреси та списку ролей. Кожна роль складається із ідентифікатора ролі та імені.

ServiceUserPostDto	{
	"password": string,
	"roleId": long,
	"username": string
	}
		Інформація для реєстрації користувача сервісу, що складається із імені користувача, паролу та ідентифікатора ролі.

ServiceUserGetDto	{ "id": long, "roleId": long, "serviceId": long, "username": string	Інформація про користувача сервісу, що складається із його імені та ідентифікаторів користувача, ролі та

Продовження таблиці 2.7

	}	сервісу.

ServiceUserInfoDto	{
	"id": long,
	"serviceId": long,
	"properties": {
	"canShareData": boolean
	},
	"queries": [
	{
	"desc": string,
	"id": long,
	"variables": [
	{
	"desc": string,
	"name": string
	}
	]
	}
	],
	"roleId": long,
	"serviceId": long,
	"tags": [
	{
	"defaultValue": string,
		Інформація про можливості користувача сервісу, що складається із індентифікаторів користувача та сервісу, списку доступних запитів, булевого значення, що вказує чи користувач може поширювати дані, а також список даних, які можна поширювати. Опис даних (тегів) та запитів аналогічний як в ServiceDefinitionGetDto.

Продовження таблиці 2.7

	"desc": string, "id": long, "name": string, "required": boolean }], "username": string }	

DataPostDto	{
	"latitude": float,
	"longitude": float,
	"tags": [
	{
	"name": string,
	"value": string
	}
	]
	}
		Інформація, яку користувач поширює в межах сервісу, що складається із географічних координат (широта і довгота) та списку тегів. Кожен тег - це пара ключ-значення.

DataRequestDto	{
	"bindingValues": [
	{
	"name": string,
	"value": string
	}
	],
	"queryId": long
		Інформація, що передається серверу для виконання запиту по даних. Вона складається із ідентифікатора запиту та списку значень змінних, що в ньому.

Продовження таблиці 2.7

	}	використовуються

Таблиця 2.8 – URI-аргументи

Назва	**Тип**	**Опис**
serviceId	long	Ідентифікатор сервісу
adminId	long	Ідентифікатор адміністратора сервісу
userId	long	Ідентифікатор користувача сервісу

2.2.3 Автентифікація

Результатом процесу автентифікації є токен доступу. При розробці платформи було прийнято видавати JWT токени, що базуються на JSON та цифровому підписі. JWT токен складається із заголовку, вмісту та сигнатури. Сигнатура обчислюється з використання симетричного алгоритму шифрування HS256, який застосовується до стрічки з заголовку та вмісту з’єднаних розділювачем. Ще одним аргументом HS256 є секрет, що й дозволяє відрізнити свою сигнатуру від чужої.

В системі існує два типи облікових записів: адміністратори та користувачі сервісів. Адміністратори відповідають за створення сервісу і описання його бізнес логіки. Користувачі сервісів існують в межах конкретного сервісу і їх можливості задаються роллю, яку створив власник (адміністратор) сервісу. Таким чином ми можемо розділити методи серверу на ті, що працюють на рівні платформи та ті, що працюють на рівні сервісу. В таблиці 2.9 наведено даний розділ методів на ті, що вимагають токен доступу адміністратора, користувача сервісу та публічно доступні.

Таблиця 2.9 – Методи по рівню доступу

Методи адміністратора	**Методи користувача сервісу**	**Публічні методи**
POST /service PUT /service DELETE /service/{serviceId} GET /service/{serviceId} GET /admin DELETE /admin	DELETE /service/{serviceId}/user GET /service/{serviceId}/user/info POST /service/{serviceId}/data POST /service/{serviceId}/post/request	POST /admin POST /service/{serviceId}/user POST /auth/login POST /service/{serviceId}/auth/login GET /service/{serviceId}/info

Оскільки у нас є дві групи методів, що мають свої особливості в автентифікації та авторизації, то й типів токенів доступу є два. Різниця між ними лежить у вмісті та секреті. Різні секрети гарантують, що методи одного типу вважатимуть токени іншого некоректними. Токен доступу користувача сервісу має такий самий вміст, що й токен адміністратора, і ще ідентифікатори сервісу та ролі. Алгоритм авторизації користувача для виконання певного методу наведений у вигляді блок-схеми на рисунку 2.4.

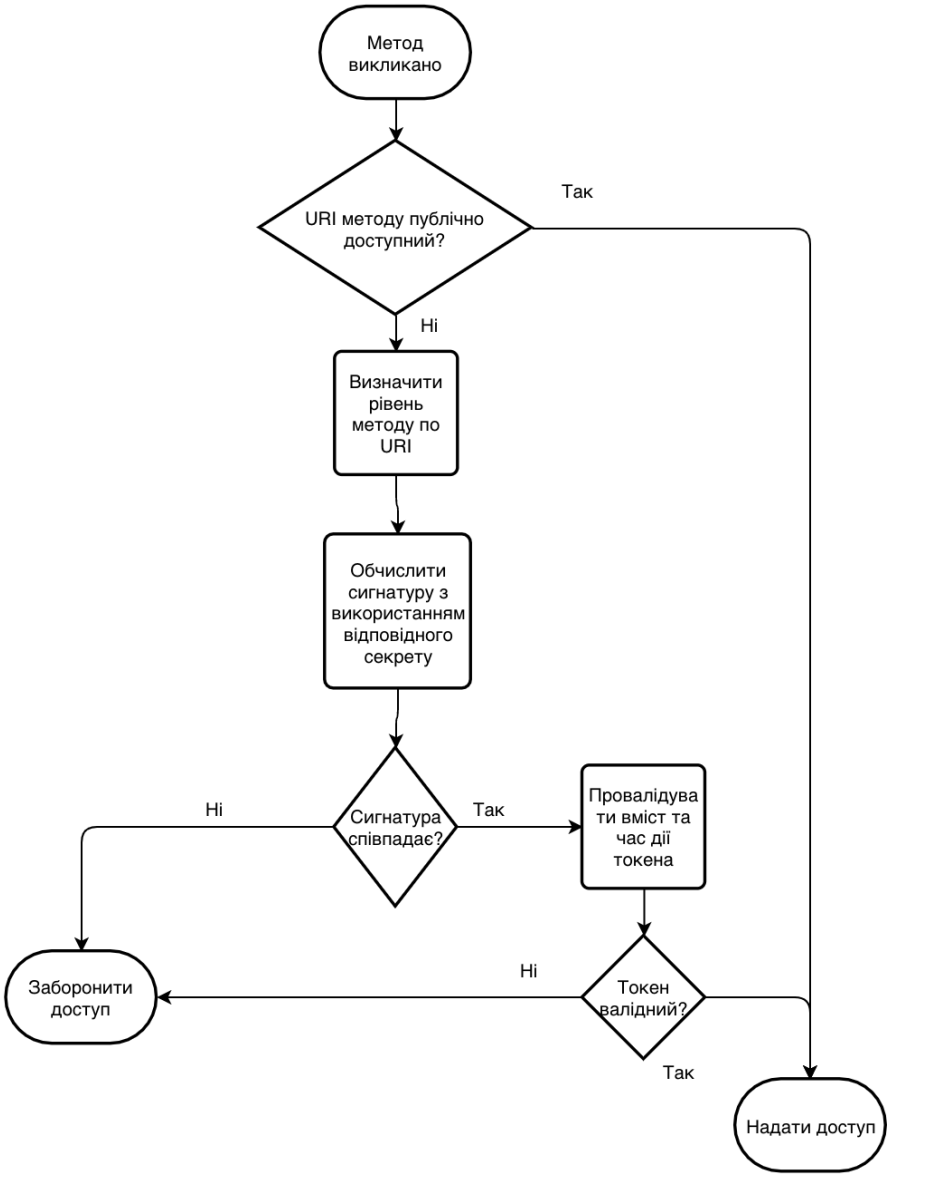

Рисунок 2.4 – Алгоритм авторизації

2.2.4 Збереження даних

Вся інформація зберігається у реляційній СКБД. Для інформації про користувачів та сервісів це типове рішення, бо об’єм невеликий і нам потрібно підтримувати повну консистентність даних. Не очевидним є рішення відносно геоданих. Ключовим фактором в його прийняті було те, що у нашій системі зберігається тільки одне (останнє) значення координати для кожного користувача, тому кількість записів в базі даних не перевищуватиме кількість користувачів. Якби нам потрібна була підтримка історії переміщень

користувача, то прийшлося б використовувати NoSQL рішення. Також використання реляційної СКБД спрощує створення власної мови запитів по геоданих і зводить її до задачі написання транслятора в SQL.

Із реляційних СКБД було обрано PostgreSQL, бо його код є з відкритим. Також важливо, що дане рішення має велику спільноту користувачів з цілого світу.

На рисунку 2.5 наведено ER-діаграма бази даних системи, а таблиці 2.10 призначення кожної з таблиць бази даних.

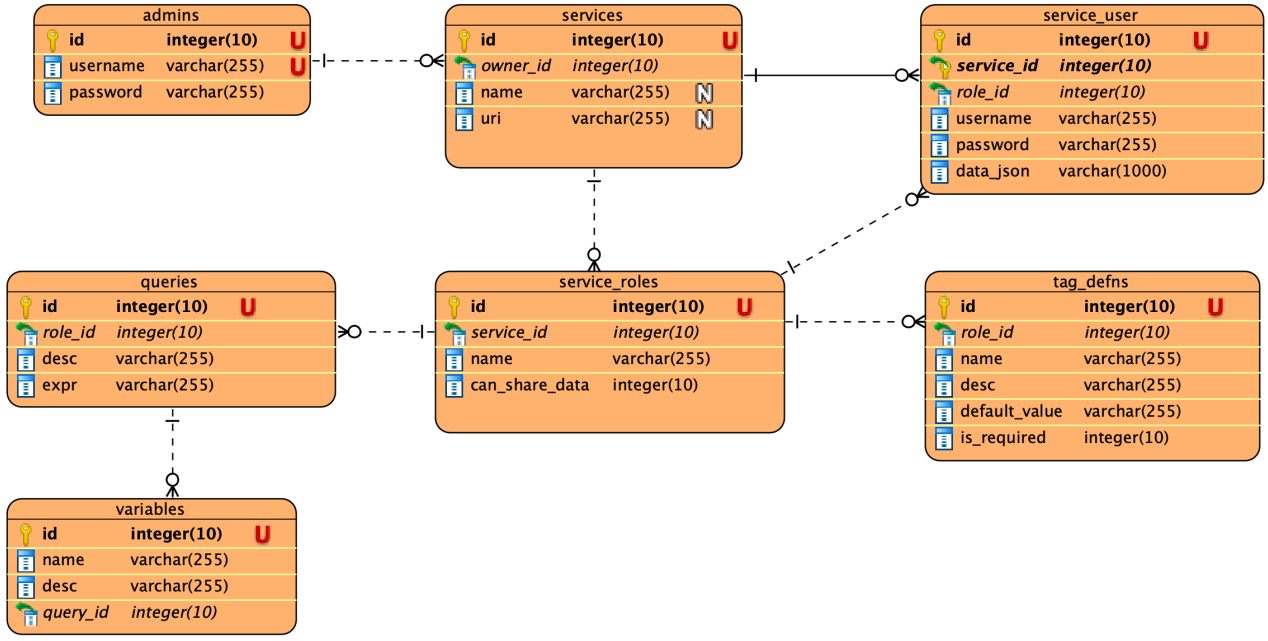

Рисунок 2.5 – ER-діаграма бази даних

Таблиця 2.10 – Призначення таблиць в базі

Назва	**Призначення**
admins	Зберігати облікові записи адміністраторів
services	Зберігати опис (бізнес-логіку) сервісів
service_roles	Зберігати опис ролей доступних для кожного сервіса

Продовження таблиці 2.10

queries	Зберігати список запитів доступних для конкретної ролі в межах сервісу
variables	Зберігати список змінних, що необхідно передати для виконання запиту
tag_defns	Зберігати список тегів, що можна поширювати разом із геоданими для конкретної ролі в межах сервісу
service_users	Зберігати облікові записи користувачів для кожного сервіс

У програмному коді структура таблиць відповідає класам із модуля com.acarus.geo.model. Завдяки модулю Data фреймворка Spring робота із таблицями замінюється на роботу з об’єктами класів. В таблиці 2.11 наведено список класів та їх опис. Анотація «@Id» вказує, що дане поле відповідає первинному ключу таблиці в базі даних. Анотація «@OneToMany» та «@ManyToOne» відображають зв’язки між таблицями. Назви відповідних полів класу і таблиці співпадають.

Таблиця 2.11 – Структура об’єктів бази даних

Назва	**Структура**	**Таблиця**
Query	public class Query { @Id @GeneratedValue(strategy = GenerationType.AUTO) private Long id;	queries

Продовження таблиці 2.11

	private String desc; private String expr; @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true, fetch = FetchType.EAGER) @PrimaryKeyJoinColumn private Set<Variable> variables; }	
Service	public class Service { @Id @GeneratedValue(strategy = GenerationType.AUTO) private Long id; @ManyToOne(fetch = FetchType.EAGER) @JoinColumn(name = "owner_id") private User owner; @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true, fetch = FetchType.EAGER) private Set<ServiceRole> roles;	services

Продовження таблиці 2.11

	private String name; private String uri; }	
ServiceRole	public class ServiceRole { @Id @GeneratedValue(strategy = GenerationType.AUTO) private Long id; @ManyToOne(fetch = FetchType.LAZY) @JoinColumn(name = "service_id") private Service service; private String name; private boolean canShareData; @OneToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL, orphanRemoval = true)	service_roles

Продовження таблиці 2.11

	private Set<Query> queries; @OneToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL, orphanRemoval = true) private Set<TagDefn> tagDefns; }	
ServiceUser	@Table(name = "service_users") public class ServiceUser implements UserDetails { @Id @GeneratedValue(strategy = GenerationType.AUTO) private Long id; @Id private Long serviceId; @ManyToOne(fetch = FetchType.EAGER) @JoinColumn(name = "role_id") private ServiceRole role;	service_users

Продовження таблиці 2.11

	private String username; private String password; }	

TagDefn	@Table(name = "tag_defns")
	public class TagDefn {
	
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long id;
	
	private String desc;
	
	private String name;
	
	private String defaultValue;
	
	private boolean isRequired;
	}
		tag_defns

Tags	@Table(name = "tags") public class Tags { @Id @GeneratedValue(strategy = GenerationType.AUTO)	tags

Продовження таблиці 2.11

	private Long id; @ManyToOne(fetch = FetchType.LAZY) private ServiceUser user; private Long serviceId; private Long timestamp; private String data; }	

Admin	@Table(name = "admins")
	public class Admin implements UserDetails {
	
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long id;
	
	private String username;
	
	private String password;
	}
		admins

Продовження таблиці 2.11

Variable	@Table(name = "variables")
	public class Variable {
	
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long id;
	
	private String name;
	
	private String desc;
	}
		variables

2.2.5 Мова запитів

Із попередніх розділів ми знаємо, що можливості користувача сервісу задаються на рівні ролі у вигляді двох списків. Перший - дані, які користувач може поширювати, другий - запитів, які користувач може виконувати. В цьому розділі ми опишемо, чим саме є запит.

Запит - предикат, який виконується на геоданих точки і її тегах. Результатом запиту є всі так точки для яких предикат повертає істину. Мова, яка використовується для написання предикату підтримує бінарні оператори, як-от EQ (рівно), NE (нерівно), LT (менше) та GT (більше). Лівим операндом завжди є назва тегу чи координати, а правим — константа чи змінна. Назва змінної починається на символ «%». Для виконання запиту користувач повинен надати значення всім змінним, що використовуються в запиті. Вирази з операторів можна комбінувати із використанням логічних операцій AND (кон'юнкція) та

OR (Диз'юнкція). Кожен вираз з оператором перед групуванням повинен бути обгорнутий у круглі дужки. Наведемо декілька прикладів предикатів.

Наприклад 1 – «(city EQ Kyiv) and (status NE Busy)». Даний запит складається із двох виразів. Перший – порівнює на рівність поле «city» з метаданих точки із літералом «Kyiv». Другий – перевіряє, що в метаданих є поле «status» і воно рівне «Busy». Подібний запит повертає список всіх користувачів, що поширюють свої дані в межах сервісу, знаходяться у Києві та не є зайнятими. Приклад 2 – «(latitude GT %minLatitude) AND (latitude % maxLatitude) AND (longtitude GT %minLongtitude) AND (longtitude LT % maxLongtitude)». Цей предикат повертає всіх користувачів сервісу, що знаходяться у заданій прямокутній області координат. Для виконання предикату користувач задає мінімальні та максимальні значення для широти та довготи. Цей запит виконується на всіх користувачах, що поширюються дані, бо координати є обов’язковими полями.

2.2.6 Мобільний застосунок

Головною задумкою при розробці мобільного застосунку була універсальність. Він повинен бути один для всіх сервісів платформи, що дозволить почати користуватися сервісом одразу ж після того як його бізнес-логіку була описана на сервері. Застосунок працює на рівні сервісу і є інструментом, який дозволяє ним користуватися. Застосунок було розроблено для платформи Android, оскільки це найбільш поширена платформа. Очікується версія ОС 4.1 та вище.

Activity — одна річ, яку може робити користувач в застосунку. Майже всі Activity створюється власне вікно в якому розміщується інтерфейс для взаємодії із користувачем. Щоб описати базовий функціонал розробленого застосунку наводимо таблицю 2.12 із основними Activity.

Таблиця 2.12 – Основні Activity застосунку

Назва	**Опис**
ServiceSelectionActivity	Дозволяє користувачу вибрати сервіс із існуючих в платформі
RegisterActivity	Дозволяє зареєструвати нового користувача у обраному сервісі. Список доступних ролей завантажується із сервера
LoginActivity	Дозволяє автентифікуватися у обраному сервісі
MainActivity	Основне вікно для користувача. Відображає можливості для його ролі. Якщо можна поширювати дані, то буде вкладка«Share». Також якщо доступні запити, то вони будуть відображені у розділі «Query» у вигляді списку
TagActivity	Дозволяє включити поширення конкретного тегу і встановити його значення
QueryActivity	Дозволяє активувати конкретний запит, встановивши значення усіх змінних, що у ньому використовуються
MapActivity	Відображає карту із результатами активованих запитів

Для взаємодії із сервером використовувалася бібліотека Volley. Серверна взаємодія написана у неблокуючому стилі з використанням асинхронних функцій і колбеків (callbacks).

2.3 Конструювання програмного забезпечення

Серверна частина розроблена на платформі Java версії 8 і поширюється у вигляді .jar файлу. Розробка виконується із використанням фреймворку Spring версії 2.1. Веб-сервер вбудований у .jar файл, що дозволяє легко тестувати та поширювати серверну частину.

Мобільний застосунок розробляється під платформа Android за допомогою середовища розробки Android Studio.

2.4 Аналіз безпеки даних

Паролі користувачів зберігаються в базі даних у хешованому вигляді. Для обчислення хешу використовується адаптивна криптографічна функція формування ключа bcrypt. Вона добре себе зарекомендувала на ринку і широко використовується. В програмному коді використовувалося її реалізація у вигляді BCryptPasswordEncoder із модуля Security фреймворку Spring.

При виконанні запиту по даних, розглядаються тільки точки із сервісу, якому належить запит, що унеможливлює написання предиката, який би міг отримати чужі дані.

2.5 Висновки по розділу

У даному розділі було описано програмний інтерфейс сервера і як взаємодіяти з ним. Проаналізувавши вимоги до авторизації та автентифікації, описано методи вирішення цих задач у даній платформі із наведенням блок-схеми. Було розкрито питання місця і формату збереження даних із наведенням ER-діаграми бази даних та опису функціональної цілі кожної таблиці. Також було розкрито питання безпечного збереження даних в системі. Після детального опису серверної частин було показано як мобільний застосунок взаємодіє із ним, надаючи універсальний інструмент для використання всіх сервісів платформи.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Для ефективної розробки і супроводу ПЗ потрібно вміти аналізувати його якість. Суть аналізу полягає у визначенні відповідності отриманого продукту з певним набором вимог. Вимоги можна класифікувати за атрибутом на який вони накладаються. Основні атрибути:

- функціональність;
- зручність;
- продуктивність;
- підтримуваність;
- надійність;
- реєстрація користувача в межах сервісу;
- авторизація користувача в межах сервісу;

Висока відповідність функціональним вимогам є основним показником того, що ПЗ готове до впровадження.

3.2 Опис процесів тестування

Метою тестування платформи є перевірка відповідності розробленого ПЗ із функціональними вимогами, поставленими на етапі проектування. Функції ПЗ, що перевіряються:

- реєстрація облікового запису адміністратора;
- авторизація адміністратора в системі;
- створення сервісу в системі;
- отримання опису сервісу;
- підтвердження бронювання адміністратором;
- реєстрація користувача в межах сервісу;
- авторизація користувача в межах сервісу;

– отримання опису можливостей доступних користувачу в межах сервісу;

– поширення геоданих користувачем;

– виконання запитів користувачем для отримання геоданих.

3.3 Опис контрольного прикладу

В таблицях 3.1 — 3.10 наведено інформацію відносно основних варіантів використання.

Таблиця 3.1 – Перевірка реєстрації адміністратора

Мета тесту	Перевірка можливості реєстрації адміністратора
Початковий стан	Відкритий метод реєстрації адміністратора в графічному клієнті Swagger UI
Вхідні дані	Параметри користувача: username, password
Схема проведення тесту	Натиснути на кнопку «Execute». Ввести вхідні дані (username, password) в формі і натиснути на кнопку «Execute».
Очікуваний результат	Створено новий обліковий запис адміністратора, інформація про наданий йому ідентифікатор відображена.
Стан програмного продукту після проведення випробувань	Створено новий обліковий запис адміністратора, надано ідентифікатор і збережено до сховища даних. У графічному клієнті відображена вся інформація про користувача.

Таблиця 3.2 – Перевірка авторизації адміністратора

Мета тесту	Перевірка можливості авторизації адміністратора у системі
Початковий стан	Відкритий метод авторизації адміністратора в графічному клієнті Swagger UI
Вхідні дані	Параметри користувача: username, password
Схема проведення тесту	Ввести вхідні дані (username, password) в формі і натиснути на кнопку «Execute».

Продовження таблиці 3.2

Очікуваний результат	Адміністратор авторизовано
Стан програмного продукту після проведення випробувань	Адміністратор авторизований. Відображений токен доступу у результатах виконання.

Таблиця 3.3 – Перевірка можливості створення сервісу

Мета тесту	Перевірка можливості створення нового сервісу в системі
Початковий стан	Відкритий метод сворення сервісу в графічному клієнті Swagger UI
Вхідні дані	Параметри сервісу: name, uri, roles.
Схема проведення тесту	Ввести вхідні дані (name, uri, roles) в форму і натиснути на кнопку «Execute».
Очікуваний результат	Сервіс створено
Стан програмного продукту після проведення випробувань	Сервіс створено. Відображений опис сервісу із присвоєними ідентифікаторами у результатах виконання.

Таблиця 3.4 – Перевірка можливості обрати сервіс

Мета тесту	Перевірка можливості обрати сервіс в мобільному застосунку.
Початковий стан	Відкритий мобільний застосунок на початковому етапі вибору сервісу.
Вхідні дані	Параметри сервісу: uri.
Схема проведення тесту	Ввести вхідні дані (uri) в поле і натиснути на кнопку «Select».
Очікуваний результат	Обрано сервіс.
Стан програмного продукту після проведення випробувань	Обрано сервіс. Відображено вікно для авторизації/реєстрації користувача в межах обраного сервісу.

Таблиця 3.5 – Перевірка можливості реєстрації користувача

Мета тесту	Перевірка можливості реєстрації облікового запису користувача в межах певного сервісу
Початковий стан	Відкритий метод реєстрації користувачів сервісу в графічному клієнті Swagger UI.

Продовження таблиці 3.5

Вхідні дані	Параметри сервісу: serviceId, roleId, username, password.
Схема проведення тесту	Ввести вхідні дані (serviceId, roleId, username, password) в форму і натиснути на кнопку «Execute».
Очікуваний результат	Створено новий обліковий запис користувача в межах сервісу, інформація про наданий йому ідентифікатор відображена.
Стан програмного продукту після проведення випробувань	Створено новий обліковий запис користувача в межах обраного сервісу, надано ідентифікатор і збережено до сховища даних. У графічному клієнті відображена вся інформація про користувача.

Таблиця 3.6 – Перевірка авторизації користувача

Мета тесту	Перевірка можливості авторизації клієнта в межах сервісу
Початковий стан	Відкритий мобільний застосунок для авторизації у обраному сервісі.
Вхідні дані	Параметри користувача: username, password
Схема проведення тесту	Ввести вхідні дані (username, password) в формі і натиснути на кнопку «Log in».
Очікуваний результат	Користувач авторизовано.
Стан програмного продукту після проведення випробувань	Користувач авторизований. У поточному вікні мобільного застосунку відображено список можливих дій.

Таблиця 3.7 – Перевірка можливості поширювати геодані

Мета тесту	Перевірка можливості користувача поширювати геодані.
Початковий стан	Відкрито розділ поширення даних основного вікна для авторизованого користувача в мобільному застосунку.
Вхідні дані	Дані: name, value.
Схема проведення тесту	Ввести дані (name, value), що поширюються разом із координатою, та натиснути на «Share».
Очікуваний результат	Дані відправляються на сервер .

Продовження таблиці 3.7

Стан програмного продукту після проведення випробувань	Дані відправляються на сервер. У мобільному застосунку немає жодних повідомлень про помилки.

Таблиця 3.8 – Перевірка можливості виконувати запити по геодані

Мета тесту	Перевірка можливості користувача виконувати запити по геодані в межах сервісу.
Початковий стан	Відкрито розділ запитів основного вікна для авторизованого користувача в мобільному застосунку.
Вхідні дані	Назва запиту, значення змінних.
Схема проведення тесту	Обрати запит, ввести значення змінних та натиснути «Activate», а потім «Open Map».
Очікуваний результат	Дані отримано із сервісу.
Стан програмного продукту після проведення випробувань	Дані отримано із сервісу. Результат відображено у вигляді точок на карті.

Таблиця 3.9 – Перевірка можливості отримувати опис сервісу

Мета тесту	Перевірка можливості незареєстрованого користувача отримати опис сервісу
Початковий стан	Відкритий метод отримання публічної інформації про сервіс в графічному клієнті Swagger UI
Вхідні дані	Адреса сервісу
Схема проведення тесту	Ввести serviceUri в форму і натиснути на кнопку «Execute».
Очікуваний результат	Отримано публічну частину опису сервісу.
Стан програмного продукту після проведення випробувань	Дані отримано із сервісу. Результат відображено у графічному інтерфейсі.

Таблиця 3.10 – Перевірка можливості отримувати опис користувача

Мета тесту	Перевірка можливості користувача отримати опис функціоналу доступного йому в межах конкретного сервісу
Початковий стан	Відкритий метод отримання інформації про автентифікованого в графічному клієнті Swagger UI, отриманий токен доступу
Вхідні дані	Ідентифікатор сервісу та токен доступу
Схема проведення тесту	Ввести serviceUri та Authorization в форму і натиснути на кнопку «Execute».
Очікуваний результат	Отримано список доступного функціоналу.
Стан програмного продукту після проведення випробувань	Дані про автентифікованого користувача отримано. Результат відображено у графічному інтерфейсі у розділі «Responses».

3.4 Висновки до розділу

В даному розділі було описано процес аналізу та тестування ПЗ. Для основних випадків використання було наведено сценарії їх тестування.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Для розгортання серверної частини потрібно виконати наступні кроки:

– Встановити JDK і JRE версії 8.

– Встановити PostgreSQL.

– Ввести логін та пароль від бази даних у файл application.yml.

– Запустити сервер виконавши наступну команду «java –jar geo-platfrom.jar».

Мобільний застосунок встановлюється вручну за допомогою файлового менеджера.

Рекомендується завантаження JDK та JRE із офіційного веб-сайту корпорації Oracle. На рисунку 4.1 показано список доступних платформ і видів інсталяторів.

Java SE Runtime Environment 8u211

You must accept the Oracle Technology Network License Agreement for Oracle Java SE to download this software.

○ **Accept License Agreement** ● **Decline License Agreement**

Product / File Description	File Size	Download
Linux x86	71.22 MB	jre-8u211-linux-i586.rpm
Linux x86	86.97 MB	jre-8u211-linux-i586.tar.gz
Linux x64	67.99 MB	jre-8u211-linux-x64.rpm
Linux x64	83.79 MB	jre-8u211-linux-x64.tar.gz
Mac OS X x64	79.36 MB	jre-8u211-macosx-x64.dmg
Mac OS X x64	70.93 MB	jre-8u211-macosx-x64.tar.gz
Solaris SPARC 64-bit	52.14 MB	jre-8u211-solaris-sparcv9.tar.gz
Solaris x64	49.93 MB	jre-8u211-solaris-x64.tar.gz
Windows x86 Online	1.95 MB	jre-8u211-windows-i586-iftw.exe
Windows x86 Offline	66.37 MB	jre-8u211-windows-i586.exe
Windows x86	68.78 MB	jre-8u211-windows-i586.tar.gz
Windows x64	76.03 MB	jre-8u211-windows-x64.exe
Windows x64	75.05 MB	jre-8u211-windows-x64.tar.gz

Рисунок 4.1 – Список доступних інсталяторів JRE 8

Якщо у вас уже встановлена інша версія Java, то потрібно зробити версію 8 версією по замовчуванню. Для початку перевірте поточну версію по замовчуванню за допомого команди «java –version». Очікуваний результат виконання команди зображений на рисунку 4.2. Якщо версія не відповідає тій, що зображена на рисунку 4.2, то потрібно її встановити вручну. Це можна зробити за допомогою вказання шляху до JRE у змінній оточення «JAVA_HOME».

```
Olegs-MacBook-Pro:~ acarus$ java -version
java version "1.8.0_201"
Java(TM) SE Runtime Environment (build 1.8.0_201-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode)
```

Рисунок 4.2 – Перевірки версії JRE по замовчуванню

По замовчуванню сервер піднімається на порту 8090. Це можна змінити вказавши інший порт у змінній server.port. Також доступні для конфігурації час життя токенів доступу та секрет, що використовується для обчислення підпису. Список конфігурацій серверу показано на рисунку 4.3.

```
security.jwt.tokenSecretKey=someSecret
security.jwt.serviceTokenSecretKey=someSecret2
security.jwt.tokenTTLSecs=18000
server.port=8090
```

Рисунок 4.3 – Список доступних інсталяторів JRE 8

Також часто виникає потреба, щоб на самому початку після розгортання сервісу уже існували якісь сервіси та користувачі. Це може бути корисно для створення супер адміністратора чи облікового запису для тестувальників. Це можна досягнути додавши свою логіку створення об’єктів в системі у метод init() класу com.acarus.geo.init.Initializer. На рисунку 4.4 зображено ініціалізація системи із сервісом доставки їжі. Він містить роль кур’єра та покупця. Для покупця доступний запит отримання замовлення по його номеру.

```
@PostConstruct
public void init() throws JsonProcessingException {
    UserGetDto userPostDto = userService.create(UserPostDto.builder().username("admin").password("admin123").

    Set<VariableDto> vars = new HashSet<>();
    vars.add(new VariableDto( name: "orderNo",  desc: "Order number"));

    QueryPostDto query = new QueryPostDto();
    query.setExpr("orderNo EQ %orderNo");
    query.setVariables(vars);
    query.setDesc("Get courier by order #");

    Set<QueryPostDto> queries = new HashSet<>();
    queries.add(query);


    Set<TagDto> tags = new HashSet<>();
    tags.add(new TagDto( name: "tel",  desc: "Tel number",  defaultValue: "",  isRequired: true));
    tags.add(new TagDto( name: "name",  desc: "Name",  defaultValue: "",  isRequired: true));
    tags.add(new TagDto( name: "orderNo",  desc: "Order number",  defaultValue: "",  isRequired: true));

    Service toCreate = ServiceDefinitionPostDto.builder()
            .name("delivery app")
            .uri("delivery")
            .roles(Arrays.asList(
                    ServiceRolePostDto.builder()
                            .name("courier")
                            .queries(queries)
                            .properties(ServiceRolePropertiesDto.builder().canShareData(true).build())
                            .tags(tags)
                            .build(),

                    ServiceRolePostDto.builder()
                            .name("customer")
                            .build()
            ))
            .build().toModel();

    toCreate.setOwner(userRepository.findById(userPostDto.getId()).get());
    toCreate.getRoles().forEach(r -> r.setService(toCreate));
    Service service = serviceRepository.save(toCreate);

    long courierRoleId = service.getRoles().stream().filter(r -> "courier".equals(r.getName())).findFirst().g
    serviceUserService.create(service.getId(), ServiceUserPostDto.builder()
            .username("oleh")
            .password("pass123")
            .roleId(courierRoleId)
            .build());
```

Рисунок 4.4 – Ініціалізація сервісу даними при старті системи

Після модифікації програмного коду його потрібно зібрати заново за допомогою утиліти maven. Для цього потрібно виконати команду «mvn clean install», знаходячись у кореневій директорії серверного коду. Рекомендована версія для використання 3.6.

4.2 Робота з програмним забезпеченням

Робота з програмним забезпеченням описана в додатках В («Керівництво адміністратора») та Г («Керівництво користувача»).

4.3 Висновки до розділу

В даному розділі було наведено інструкцію розгортання та роботи для серверної частини та мобільного застосунку.

ВИСНОВКИ

Під час виконання дипломної роботи було вивчено та описано предметну область застосунків з обміну геоданими, проаналізовано відомі програмні продукти і показано актуальність розробки платформи для спрощення процесу створення подібних застосунків.

Спроектована платформа, що складається із серверної частини та мобільного застосунку. Серверна частина надає програмний інтерфейс створення та керування сервісами, а також графічний інтерфейс для взаємодії з ним. Для взаємодії користувачів із сервісом було розроблено мобільний застосунок, що дозволяє надсилати, отримувати та візуалізувати дані. Розроблений мобільний застосунок є універсальним і використовується для всіх сервісів платформи.

Після розробки програмного забезпечення було розроблено і виконано план його тестування, що гарантує високу якість і надійність роботи.

До програмного продукту було розроблено детальну інструкцію по розгортанню та роботі із ним для адміністратора сервісу. Також було розроблено інструкцію користування мобільним застосунком.

Отже, результатом даної роботи є платформа, що значно спрощує процес створення і підтримки застосунків, де основна бізнес-логіка зосереджена на зборі та обробці геоданих. Платформа значно знижує поріг входу для IT-підприємців, що має позитивний вплив на кількість та якість подібних програмних продуктів на ринку.

ПЕРЕЛІК ПОСИЛАНЬ

1) McMillan C. Attendify API Helps Developers Create Event Apps [Електронний ресурс] / Candice McMillan. – 2015. – Режим доступу до ресурсу: http://www.softline.kiev.ua/ua/khmarni-poslugi/poslugi-ukhmarakh/platforma-yak-posluga.html.

2) Mernik M. When and How to Develop Domain-Specific Languages / M. Mernik, A. Sloane, J. Heering // ACM Computing Surveys / M. Mernik, A. Sloane, J. Heering., 2005. – С. 316–344.

3) Fox A. Engineering Software as a Service: An Agile Approach Using Cloud Computing / A. Fox, D. Patterson., 2013. – 474 с.

4) Introduction to JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: https://jwt.io/introduction/.

5) Milosevic D. REST Security with JWT using Java and Spring Security [Електронний ресурс] / Dejan Milosevic – Режим доступу до ресурсу: https://www.toptal.com/java/rest-security-with-jwt-spring-security-and-java.

6) Tyagi V. Volley Library in Android [Електронний ресурс] / Vartika Tyagi – Режим доступу до ресурсу: https://www.geeksforgeeks.org/volley-library-in-android/.

7) Spring Boot - Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm.

8) Коваль Г.И., Коротун Т.М., Остапенко А.П. Превентивное тестирование и оценка надежности программного обеспечения как форма 49 управления риском проекта. //Сб. Программная инженерия. – Киев., 1993. – С. 19-26.

9) Skiena S. The Algorithm Design Manual / Steven Skiena., 2008. – 748 с.

10) Szczukocki D. Interpreter Design Pattern in Java [Електронний ресурс] / Denis Szczukocki // Baeldung. – 2018. – Режим доступу до ресурсу: https://www.baeldung.com/java-interpreter-pattern.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ О.А. Павлов

“___” _____ 2019 р.

Платформа для побудови застосунків з обміну геоданими

Додаток А Технічне завдання

КПІ.ІП-5107.045440-03-91

“ПОГОДЖЕНО”

Керівник проекту:

_____ О.К. Очеретяний

Виконавець:

_____ О.В. Кліщ

Нормоконтроль:

_____ К.І. Ліщук

Київ – 2019 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Платформа для побудови застосунків з обміну геоданими

Галузь застосування: Інформаційно-комунікаційні системи

Наведене технічне завдання поширюється на розробку платформи для побудови застосунків з обміну геоданими [045430], котра використовується для створення сервісів в яких центральну роль відіграє робота з географічними координатами клієнтів та призначена для користувачів, що хочуть у короткий термін та без знань програмування створити власний продукт в якому бізнес-логіка зосереджена на зборі та обробці геоданих.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки платформи для побудови застосунків з обміну геоданими є завдання на дипломне проектування, затверджене кафедрою автоматизованих систем обробки інформації і управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім.Ігоря Сікорського).

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для користувачів, що хочуть створити сервіс в якому бізнес-логіка зосереджена на зборі та обробці геоданих, але не мають достатню кількість ресурсів для написання програмного продукту самому.

Метою розробки є платформа, що дозволить конструювати сервіси із власною бізнес-логікою збору та обробки геоданих у короткий термін та без написання програмного коду.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1 Для користувача:

- вибір сервісу;
- реєстрація в сервісі;
- авторизація;
- перегляд дій дозволених для почної ролі;
- поширення геоданих;
- перегляд списку метаданих дозволених для поширення;
- поширення метаданих разом із координатою;
- перегляд списку запитів;
- виконання запиту;
- перегляд результату запитів на карті;

4.1.1.2 Для адміністратора системи:

- реєстрація;
- авторизація;
- створіення сервісу;
- редагування існуючого сервісу;
- створення ролей в сервісі;
- опис запитів доступних для певної ролі;
- опис даних доступних для поширення для певної ролі.

4.1.2 Розробку виконати на платформі Linux.

4.1.3 Додаткові вимоги:

- дані різних сервісів повинні бути ізольовані одне від одного;
- спосіб опису бізнес-логіки сервісу повинен бути декларативним.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації.

4.2.2 Передбачити захист від некоректних дій користувача.

4.2.3 Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.2 Обслуговування

Обслуговування платформи проводиться системним адміністратором, який також відповідає за розгортання та налаштування системи.

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1 Серверне програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

4.4.2 Мінімальна конфігурація технічних засобів:

Тип процесору - Intel Core x64.

Об‘єм ОЗП - 8 Гб.

4.5 Вимоги до інформаційної та програмної сумісності

4.5.1 Сервере програмне забезпечення повинно працювати під управлінням операційних систем сімейства WIN32 (Windows'XP, Windows NT і т.д.) або Unix. Клієнтське програмне забезпечення повинно працювати під управлінням операційної системи Android.

4.5.2 Вхідні дані повинні бути представлені в наступному форматі: дані, що передаються клієнтом серверу має бути у форматі JSON (JavaScript Object Notation).

4.5.3 Результати повинні бути представлені в наступному форматі: результати, що передається від сервера клієнту мають бути у форматі JSON (JavaScript Object Notation).

4.5.4 Серверне програмне забезпечення повинно надавати REST (Representational State Transfer) інтерфейс для потенційної інтеграції із стороннім програмним забезпечення.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8 Спеціальні вимоги

Згенерувати установчу версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

5.2 Програмне забезпечення повинно мати довідникову систему

5.3 У склад супроводжувальної документації повинні входити наступні документи:

5.3.1 Пояснювальна записка не менше ніж на 50 аркушах формату А4 (без додатків 5.3.2 - 5.3.4).

5.3.2 Технічне завдання.

5.3.3 Керівництво користувача.

5.3.4 Програма та методика тестування

5.4 Графічна частина повинна бути виконана на аркушах формату А3, котрі включаються у якості додатків до пояснювальної записки:

5.4.1 Схема структурна компонентів структур даних

5.4.2 Схема структурна варіантів використання

5.4.3 Схема структурна класів програмного забезпечення

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	20.04.2019	
2.	Розробка технічного завдання	23.05.2019	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	25.04.2019	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	11.05.2019	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму …)
5.	Програмна реалізація програмного забезпечення	16.06.2019	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	20.05.2019	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	25.05.2019	Пояснювальна записка.
8.	Розробка матеріалів графічної частини проекту	25.05.2019	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	27.05.2019	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

7.1 Види випробувань

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ О.А. Павлов

“___” _____ 2019 р.

Платформа для побудови застосунків з обміну геоданими

Додаток Б Програма та методика тестування

КПІ.ІП-5107.045430.04.51

“ПОГОДЖЕНО”

Керівник проекту:

_____ О.К. Очеретяний

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____О.В. Кліщ

Київ – 2019 року

1 ОБ'ЄКТ ВИПРОБУВАНЬ

Об'єктом випробувань є платформа для побудови застосунків з обміну геоданими, що дозволяє створити їх описавши лише бізнес-логіку у декларативному стилі.

2 МЕТА ТЕСТУВАННЯ

Процес тестування має на меті наступні цілі:

- перевірка на відповідність функціональним вимогам;
- перевірка на відповідність нефункціональним вимогам;
- пошук несистемних збоїв платформи.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування платформи набір методів:

- ручне тестування коду та інтерфейсу;
- автоматизоване тестування коду;

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Перевірка роботи платформи відбувається за допомогою наступних методів:

- ручне тестування на коректних вхідних даних;
- ручне тестування на некоректних вхідних даних;
- ручне тестування застосунку на граничних значеннях;
- автоматизоване тестування на коректних вхідних даних;
- автоматизоване тестування на некоректних вхідних даних;
- автоматизоване тестування на граничних значеннях;
- автоматизоване тестування за стосунку для коректних вхідних даних;
- автоматичне стрес-тестування.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ О.А. Павлов

“___” _____ 2019 р.

Платформа для побудови застосунків з обміну геоданими

Додаток В Керівництво адміністратора

КПІ.ІП-5107.045430.05.51

“ПОГОДЖЕНО”

Керівник проекту:

_____ О.К. Очеретяний

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____О.В. Кліщ

Київ – 2019 року

Виклик API методів

Для початку роботи потрібно відкрити Swagger UI, що знаходиться по адресі /swagger-ui.html#. На рисунку Б.1 зображено екран із списком методів доступних системі.

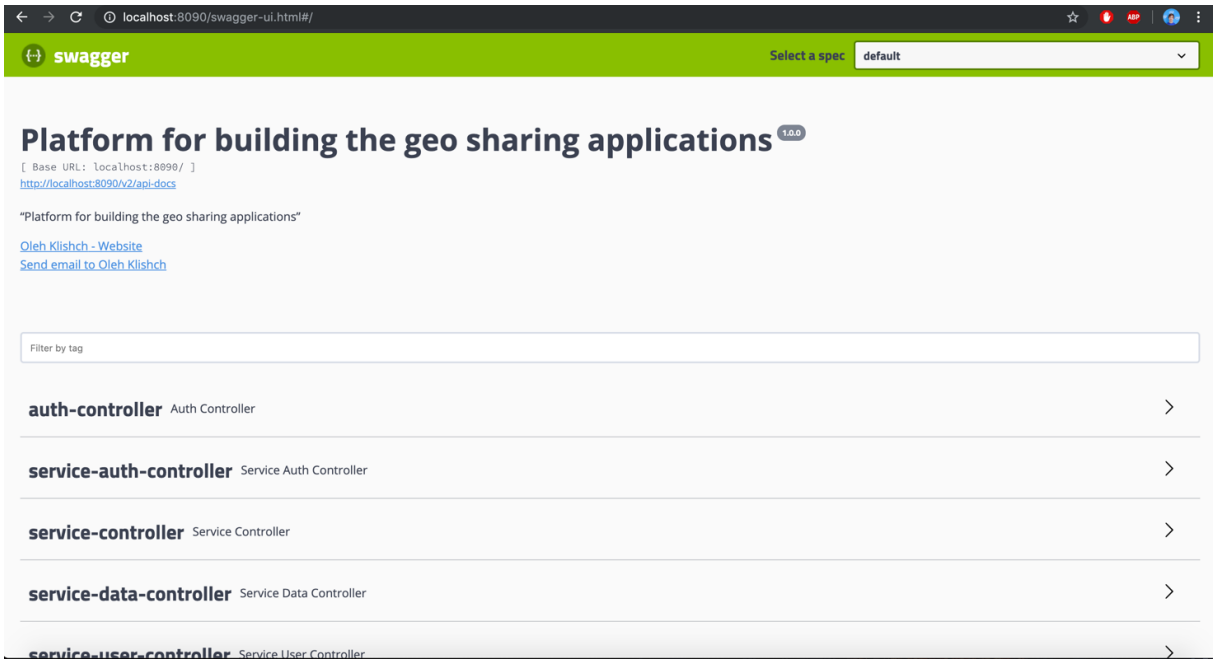

Рисунок В.1 Графічний інтерфейс для роботи із API

Для взаємодії із сервером потрібно натиснути на один з доступних методів. Далі потрібно ввести тіло запиту у форматі JSON, інші параметри запиту та натиснути «Execute». Вікно для виклику методу зображено на риснку В.2. Для методів, що вимагають токен доступу потрібно вказати його у полі заголовку «Authorization», додавши префікс «Bearer ».

Рисунок В.2 Вікно виклику методу

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ О.А. Павлов

“___” _____ 2019 р.

Платформа для побудови застосунків з обміну геоданими

Додаток Г Керівництво користувача

КПІ.ІП-5107.045440-06-34

“ПОГОДЖЕНО”

Керівник проекту:

_____ О.К. Очеретяний

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____ О.В. Кліщ

Київ – 2019

ВЗАЄМОДІЯ ІЗ ПЛАТФОРМОЮ

Для взаємодії із сервісом потрібно відкрити мобільний застосунок. Першим кроком є вибір сервісу. Для цього потрібно ввести назву сервісу у поле та натиснути «Select». На рисунку Г.1 зображено вікно вибору сервісу.

Рисунок Г.1 Вікно вибору сервісу

Після вибору сервісу відобразиться вікно авторизації. Якщо у вас вже є наявний обліковий запис, то достатньо ввести логін, пароль та натиснути «Sign In». На рисунку Г.2 зображено вікно авторизації.

Рисунок Г.2 Вікно авторизації

Якщо у вас ще немає облікового запису чи бажаєте створити новий, натисніть на «Register New». На рисунку Г.3 зображено вікно реєстрації. Для створення нового облікового запису потрібно обрати роль та ввести логін із паролем.

Рисунок Г.3 Вікно реєстрації

Після авторизації буде відображено головне вікно застосунку як на рисунку Г.4. Якщо поточно роль передбачає можливість поширювати дані, то буде показаний розділ «Share». Також якщо можна виконувати запити, то буде відображений розділ «Query».

Рисунок Г.4 Основне вікно застосунку

Щоб разом із координатою поширювати метадані, потрібно відкрити розділ кожного з тегів, ввести значення та активувати даний тег. Вікно роботи із тегом показано на рисунку Г.5. Після чого потрібно почати поширювати дані активувавши «Sharing».

Рисунок Г.5 Вікно тега

Для виконання запиту потрібно перейти у розділ «Query», який показаний на рисунку Г.6. Перейти в якийсь конкретний запит з тих, що наведено у списку доступних.

Рисунок Г.6 Розділ із списком запитів

У випадаючому списку наведено імена змінних, які потрібно встановити для виконання запиту. Встановіть їх значення та активуйте запит натиснувши на «Activate query». На рисунку Г.7 показано вікно запиту.

Рисунок Д.7 Розділ запиту

Після активації запиту мобільний застосунок почне запитувати дану у сервера та відображати результат на карті. Для перегляду карти натисніть «Open Map» у вікні із запитами. На рисунку Д.8 показано карту із результатом запиту.

Рисунок Д.8 Карта із даними

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ О.А. Павлов

“___” _____ 2019 р.

Платформа для побудови застосунків з обміну геоданими

Додаток Д Опис програми

КПІ.ІП-5107.045430.07.13

“ПОГОДЖЕНО”

Керівник проекту:

_____ О.К. Очеретяний

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____О.В. Кліщ

Київ – 2019 року

Тексти програмного коду

Платформа для побудови застосунків з обміну геоданими

(Найменування програми (документа))

DVD-R

(Вид носія даних)

144 арк, 631 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ - 2019

Опис серверної частини

```
package com.acarus.geo.controller;

import com.acarus.geo.dto.LogInRequestDto;
import com.acarus.geo.dto.LogInResponseDto;
import com.acarus.geo.service.impl.AuthServiceImpl;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.validation.Valid;

@RestController
@RequestMapping("/auth")
@RequiredArgsConstructor(onConstructor = @__(@Autowired))
public class AuthController {

   private final AuthServiceImpl authService;

   @PostMapping(value = "/login", produces = "application/json")
   public LogInResponseDto login(@RequestBody @Valid LogInRequestDto request) {
      return authService.login(request);
   }
}
```

```
package com.acarus.geo.controller;

import com.acarus.geo.dto.LogInRequestDto;
import com.acarus.geo.dto.LogInResponseDto;
import com.acarus.geo.service.impl.ServiceAuthServiceImpl;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;

@RestController
@RequestMapping("/service/{serviceId}/auth")
@RequiredArgsConstructor(onConstructor = @__(@Autowired))
public class ServiceAuthController {
```

```
    private final ServiceAuthServiceImpl authService;

    @PostMapping(value = "/login", produces = "application/json")
    public LogInResponseDto login(@PathVariable long serviceId,
                                  @RequestBody @Valid LogInRequestDto request) {
        return authService.login(serviceId, request);
    }
}
```

```
package com.acarus.geo.controller;

import com.acarus.geo.dto.ServiceDefinitionGetDto;
import com.acarus.geo.dto.ServiceDefinitionPostDto;
import com.acarus.geo.dto.ServiceInfoDto;
import com.acarus.geo.service.ServiceService;
import io.swagger.annotations.ApiImplicitParam;
import io.swagger.annotations.ApiImplicitParams;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;

import static com.acarus.geo.constants.SwaggerConstants.JWT_TOKEN_DESC;
import static com.acarus.geo.constants.SwaggerConstants.JWT_TOKEN_HEADER_NAME;

@RestController
@RequestMapping("/service")
@RequiredArgsConstructor(onConstructor = @__(@Autowired))
public class ServiceController {

    private final ServiceService serviceService;

    @ApiImplicitParams({@ApiImplicitParam(
            name = JWT_TOKEN_HEADER_NAME,
            value = JWT_TOKEN_DESC,
            paramType = "header",
            required = true),
    })
    @PostMapping(produces = "application/json")
    public ServiceDefinitionGetDto create(@RequestBody @Valid ServiceDefinitionPostDto service) {
        return serviceService.create(service);
    }

    @GetMapping("/{serviceUri}")
    public ServiceInfoDto getServiceInfo(@PathVariable String serviceUri) {
        return serviceService.getServiceInfo(serviceUri);
    }
}
```

```
package com.acarus.geo.controller;

import com.acarus.geo.dto.DataGetDto;
import com.acarus.geo.dto.DataPostDto;
import com.acarus.geo.dto.DataRequestDto;
import com.acarus.geo.service.DataService;
import com.fasterxml.jackson.core.JsonProcessingException;
import io.swagger.annotations.ApiImplicitParam;
import io.swagger.annotations.ApiImplicitParams;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;

import java.io.IOException;
import java.util.List;

import static com.acarus.geo.constants.SwaggerConstants.JWT_TOKEN_DESC;
import static com.acarus.geo.constants.SwaggerConstants.JWT_TOKEN_HEADER_NAME;

@RestController
@RequestMapping("/service/{serviceId}/data")
@RequiredArgsConstructor(onConstructor = @__(@Autowired))
public class ServiceDataController {
   private final DataService dataService;

   @ApiImplicitParams({@ApiImplicitParam(
         name = JWT_TOKEN_HEADER_NAME,
         value = JWT_TOKEN_DESC,
         paramType = "header",
         required = true),
   })
   @PostMapping
   public void share(@PathVariable long serviceId, @RequestBody @Valid DataPostDto data) throws
JsonProcessingException {
      dataService.share(data);
   }

   @ApiImplicitParams({@ApiImplicitParam(
         name = JWT_TOKEN_HEADER_NAME,
         value = JWT_TOKEN_DESC,
         paramType = "header",
         required = true),
   })
```

```
@PostMapping("/request")
   public List<DataPostDto> fetch(@PathVariable long serviceId, @RequestBody @Valid DataRequestDto
request) throws IOException {
      return dataService.fetch(request);

}
 }
```

```
package com.acarus.geo.controller;

import com.acarus.geo.dto.ServiceUserGetDto;
import com.acarus.geo.dto.ServiceUserInfoDto;
import com.acarus.geo.dto.ServiceUserPostDto;
import com.acarus.geo.model.ServiceUser;
import com.acarus.geo.service.ServiceUserService;
import com.acarus.geo.utils.AuthenticatedUserUtils;
import io.swagger.annotations.ApiImplicitParam;
import io.swagger.annotations.ApiImplicitParams;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;

import static com.acarus.geo.constants.SwaggerConstants.JWT_TOKEN_DESC;
import static
com.acarus.geo.constants.SwaggerConstants.JWT_TOKEN_HEADER_NAME;

@RestController
@RequestMapping("/service/{serviceId}/user")
@RequiredArgsConstructor(onConstructor = @__(@Autowired))
public class ServiceUserController {

   private final ServiceUserService userService;

   @PostMapping(produces = "application/json")
   public ServiceUserGetDto register(@PathVariable long serviceId, @RequestBody
@Valid ServiceUserPostDto user) {
      return userService.create(serviceId, user);
   }
```

```
    @GetMapping(value = "/info", produces = "application/json")
    public ServiceUserInfoDto getUserInfo(@PathVariable long serviceId) {
        ServiceUser user = (ServiceUser)
AuthenticatedUserUtils.getAuthentication().getDetails();

        return ServiceUserInfoDto.fromModel(user);
    }
}
```

```
package com.acarus.geo.controller;

import com.acarus.geo.dto.UserGetDto;
import com.acarus.geo.dto.UserPostDto;
import com.acarus.geo.service.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;

@RestController
@RequestMapping("/user")
@RequiredArgsConstructor(onConstructor = @__(@Autowired))
public class UserController {

    private final UserService userService;

    @PostMapping(produces = "application/json")
    public UserGetDto register(@RequestBody @Valid UserPostDto user) {
        return userService.create(user);
    }

    @DeleteMapping("/{id}")
    public void delete(@PathVariable long id) {
        userService.delete(id);
    }
```

```
@GetMapping(value = "/{id}", produces = "application/json")
    public UserGetDto get(@PathVariable long id) {
        return userService.get(id);
    }
}
```

```
package com.acarus.geo.filter;

import java.util.Map;

public abstract class AbstractFilterToSpecificationConverter<T> {

    public abstract Object stringToValue(String fieldName, String val);

    public boolean eval(String query, Map<String, String> biddingValues,
Map<String, Object> data) {
        query = query.trim();
        if (isTerminalOperation(query)) {
            return parseTerminalOperation(query, biddingValues, data);
        }
        int balance = 1;
        int pos = 1;
        while (balance != 0 && pos < query.length()) {
            if (query.charAt(pos) == '(') {
                ++balance;
            } else if (query.charAt(pos) == ')') {
                --balance;
            }
            ++pos;
        }
        if (balance != 0) {
            throw new RuntimeException("Parenthesis doesn't match");
        }
        boolean res = eval(query.substring(1, pos - 1), biddingValues, data);
        if (pos == query.length()) {
            return res;
        }
        int next = query.indexOf('(', pos);
        if (next < 0) {
```

```
            throw new RuntimeException("Open parenthesis is missing");
        }
        String joiner = query.substring(pos, next - 1).trim().toLowerCase();
        switch (joiner) {
            case "and":
                return res && eval(query.substring(next), biddingValues, data);

            case "or":
                return res || eval(query.substring(next), biddingValues, data);

            default:
                throw new RuntimeException("Join operation: " + joiner + " isn't
supported");
        }
    }

    private boolean isTerminalOperation(String query) {
        return !query.startsWith("(");
    }

    private boolean parseTerminalOperation(String query, Map<String, String>
biddingValues, Map<String, Object> data) {
        int firstSeparator = query.indexOf(' ');
        if (firstSeparator < 0) {
            throw new RuntimeException("Condition has an invalid format");
        }

        int secondSeparator = firstSeparator;
        while (secondSeparator < query.length() && query.charAt(secondSeparator) ==
' ') {
            ++secondSeparator;
        }
        if (secondSeparator == query.length() || query.indexOf(' ', secondSeparator) < 0)
{
            throw new RuntimeException("Condition has an invalid format");
        }

        secondSeparator = query.indexOf(' ', secondSeparator);
        final String field = query.substring(0, firstSeparator).trim();
        final String op = query.substring(firstSeparator,
```

```
secondSeparator).trim().toLowerCase();
      String expr = query.substring(secondSeparator).trim();
      final String fieldValue = biddingValues.get(expr);
      if (fieldValue == null) {
         throw new RuntimeException("Var " + expr + " isn't supplied");
      }
      final Object value = stringToValue(field, fieldValue.startsWith("'") || fieldValue.startsWith("\"") ?
            fieldValue.substring(1, fieldValue.length() - 1) :
            fieldValue);

      if (!data.containsKey(field)) {
         return false;
      }

      switch (op) {
         case "eq":
            return data.get(field).equals(value);

         case "lt":
            return Double.parseDouble(data.get(field).toString()) < Double.parseDouble(value.toString());

         case "gt":
            return Double.parseDouble(data.get(field).toString()) > Double.parseDouble(value.toString());

         default:
            return false;
      }
   }
}
```

```
package com.acarus.geo.security;

import com.acarus.geo.exceptions.ErrorDto;
import com.acarus.geo.exceptions.InvalidJwtAuthenticationException;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.util.AntPathMatcher;
import org.springframework.web.filter.GenericFilterBean;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.time.Instant;

@RequiredArgsConstructor
public class JwtTokenFilter extends GenericFilterBean {

   private final AbstractTokenProvider platformTokenProvider;

   private final AbstractTokenProvider serviceTokenProvider;

   @Override
   public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain)
         throws IOException, ServletException {
      try {
         String token = AbstractTokenProvider.resolveToken((HttpServletRequest) request);
         if (token != null) {
            AbstractTokenProvider jwtTokenProvider = resolveTokenProvider(request);
            if (!jwtTokenProvider.validateToken(token)) {
               throw new InvalidJwtAuthenticationException("Expired or invalid JWT token");
            }

SecurityContextHolder.getContext().setAuthentication(jwtTokenProvider.getAuthentication(token));
         }
      } catch (Exception ex) {
         handleError((HttpServletResponse) response, ex);
         return;
      }
      filterChain.doFilter(request, response);
   }

   private AbstractTokenProvider resolveTokenProvider(ServletRequest request) {
      AntPathMatcher pathMatcher = new AntPathMatcher();
      if (pathMatcher.match("/service/{serviceId:\\d+}/{object:[a-zA-Z]+}/**", ((HttpServletRequest)
request).getRequestURI())) {
         return serviceTokenProvider;
      } else {
         return platformTokenProvider;
      }
   }

   private void handleError(HttpServletResponse response, Exception ex) throws IOException {
      response.setStatus(HttpStatus.UNAUTHORIZED.value());
      response.setContentType("application/json");
      ErrorDto error = new ErrorDto(HttpStatus.UNAUTHORIZED.value(), ex.getMessage(),
Instant.now().toEpochMilli());
```

```
        response.getWriter().print(new ObjectMapper().writeValueAsString(error));
        response.getWriter().flush();
    }
}
```

Опис мобільної частини

```
package com.acarus.geo.network;

import android.content.Context;

import com.acarus.geo.data.LoginRepository;
import com.acarus.geo.data.model.LoginCallback;
import com.android.volley.AuthFailureError;
import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.JsonObjectRequest;
import com.android.volley.toolbox.Volley;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.util.HashMap;
import java.util.Map;

public final class NetworkClient {

    private static final String BASE_URL = "http://10.0.2.2:8090";

    private static final String SERVICE_INFO_URI = BASE_URL + "/service/${serviceName}";
    private static final String USER_INFO_URI = BASE_URL + "/service/${serviceId}/user/info";
    private static final String REGISTER_USER_URI = BASE_URL + "/service/${serviceId}/user";
    private static final String LOGIN_URI = BASE_URL + "/service/${serviceId}/auth/login";
    private static final String SHARE_DATA_URI = BASE_URL + "/service/${serviceId}/data";
    private static final String REQUEST_DATA_URI = BASE_URL + "/service/${serviceId}/data/request";

    private static final String SERVICE_NAME_VAR = "${serviceName}";
    private static final String SERVICE_ID_VAR = "${serviceId}";

    private static NetworkClient instance;

    private RequestQueue requestQueue;

    private NetworkClient(Context ctx) {
        requestQueue = Volley.newRequestQueue(ctx);
    }
```

```
public static synchronized NetworkClient getInstance(Context ctx) {
    if (instance == null) {
        instance = new NetworkClient(ctx);
    }
    return instance;
}

public RequestQueue getRequestQueue() {
    return requestQueue;
}

public void getServiceInfo(String serviceName, final NetworkCallback<JSONObject> callback) {
    JsonObjectRequest request = new JsonObjectRequest(Request.Method.GET,
            SERVICE_INFO_URI.replace(SERVICE_NAME_VAR, serviceName),
            null, new Response.Listener<JSONObject>() {
        @Override
        public void onResponse(JSONObject response) {
            callback.onSuccess(response);
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            // TODO: Handle error
            callback.onFail(error.networkResponse.statusCode, new String(error.networkResponse.data));
        }
    });
    getRequestQueue().add(request);
}

public void getUserInfo(final String token, long serviceId, final NetworkCallback<JSONObject> callback) {
    JsonObjectRequest request = new JsonObjectRequest(
            Request.Method.GET,
            USER_INFO_URI.replace(SERVICE_ID_VAR, Long.toString(serviceId)),
            null,
            new Response.Listener<JSONObject>() {
                @Override
                public void onResponse(JSONObject response) {
                    LoginRepository.getInstance(null).getUser().setUserInfo(response);
                    callback.onSuccess(response);
                    System.out.println(response);
                }
            }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            // TODO: Handle error
            System.out.println(error);
            callback.onFail(error.networkResponse.statusCode, new String(error.networkResponse.data));
        }
    }) {
        @Override
        public Map getHeaders() throws AuthFailureError {
```

```
HashMap headers = new HashMap();
            headers.put("Content-Type", "application/json");
            headers.put("Authorization", "Bearer " + token);
            return headers;
        }
    };
    getRequestQueue().add(request);
  }

  public void getDataForQuery(final String token, long serviceId, long queryId, JSONArray bindingValues,
final NetworkCallback<JSONArray> callback) {
    JSONObject query = new JSONObject();
    try {
      query.put("queryId", queryId);
      query.put("bindingValues", bindingValues);
    } catch (JSONException e) {
      e.printStackTrace();
    }

    CustomRequest request = new CustomRequest(
        Request.Method.POST,
        REQUEST_DATA_URI.replace(SERVICE_ID_VAR, Long.toString(serviceId)),
        query,
        new Response.Listener<JSONArray>() {
          @Override
          public void onResponse(JSONArray response) {
            callback.onSuccess(response);
            System.out.println(response);
          }
        }, new Response.ErrorListener() {
      @Override
      public void onErrorResponse(VolleyError error) {
        // TODO: Handle error
        System.out.println(error);
        callback.onFail(error.networkResponse.statusCode, new String(error.networkResponse.data));
      }
    }) {
      @Override
      public Map getHeaders() throws AuthFailureError {
        HashMap headers = new HashMap();
        headers.put("Content-Type", "application/json");
        headers.put("Authorization", "Bearer " + token);
        return headers;
      }
    };
    getRequestQueue().add(request);
  }
```

```
    public void registerUser(long serviceId, long roleId, String username, String password, final
NetworkCallback<JSONObject> callback) {
        JSONObject userInfo = new JSONObject();
        try {
            userInfo.put("roleId", roleId);
            userInfo.put("username", username);
            userInfo.put("password", password);
        } catch (JSONException e) {
            e.printStackTrace();
        }

        JsonObjectRequest request = new JsonObjectRequest(
                Request.Method.POST,
                REGISTER_USER_URI.replace(SERVICE_ID_VAR, Long.toString(serviceId)),
                userInfo,
                new Response.Listener<JSONObject>() {
                    @Override
                    public void onResponse(JSONObject user) {
                        System.out.println("Registered");
                        callback.onSuccess(user);
                    }
                },
                new Response.ErrorListener() {
                    @Override
                    public void onErrorResponse(VolleyError error) {
                        // TODO: Handle error
                        System.out.println("Failed to register");
                        callback.onFail(error.networkResponse.statusCode, new
String(error.networkResponse.data));
                    }
                });
        getRequestQueue().add(request);
    }

    public void login(long serviceId, String username, String password, final LoginCallback callback) {
        JSONObject credentials = new JSONObject();
        try {
            credentials.put("username", username);
            credentials.put("password", password);
        } catch (JSONException e) {
            e.printStackTrace();
        }
```

```
JsonObjectRequest request = new JsonObjectRequest(Request.Method.POST,
            LOGIN_URI.replace(SERVICE_ID_VAR, Long.toString(serviceId)),
            credentials, new Response.Listener<JSONObject>() {
  @Override public void onResponse(JSONObject response) {
            try {
               callback.onSuccess(response.getString("token"), response.getLong("exp"));
            } catch (JSONException e) {
               e.printStackTrace();
            }
         }
      }, new Response.ErrorListener() {
         @Override
         public void onErrorResponse(VolleyError error) {
            // TODO: Handle error
            callback.onFail(error.networkResponse.statusCode, new String(error.networkResponse.data));
         }
      });
      getRequestQueue().add(request);
   }

   public void shareData(JSONObject data, long serviceId, final String token) {
      JsonObjectRequest request = new JsonObjectRequest(Request.Method.POST,
            SHARE_DATA_URI.replace(SERVICE_ID_VAR, Long.toString(serviceId)),
            data, new Response.Listener<JSONObject>() {
         @Override
         public void onResponse(JSONObject response) {
//             System.out.println("Got response: " + response);
         }
      }, new Response.ErrorListener() {
         @Override
         public void onErrorResponse(VolleyError error) {
            // TODO: Handle error
//             System.out.println("Got error: " + error);
         }
      }) {
         @Override
         public Map getHeaders() throws AuthFailureError {
            HashMap headers = new HashMap();
            headers.put("Content-Type", "application/json");
            headers.put("Authorization", "Bearer " + token);
            return headers;
         }
      };
      getRequestQueue().add(request);
   }
}
```

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ О.А. Павлов

“___” _____ 2019 р.

Платформа для побудови застосунків з обміну геоданими

Додаток Є Графічні матеріали

КПІ.ІП-5107.045430.08.99

ПОГОДЖЕНО”

Керівник проекту:

_____ О.К. Очеретяний

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____О.В. Кліщ

Київ – 2019 року